



Credit approval by a clustering genetic algorithm

E. R. Hruschka^{1,2} & N. F. F. Ebecken²

¹*UNICENP - Positivo University Center, Brazil.*

²*COPPE / Federal University of Rio de Janeiro, Brazil.*

Abstract

This paper presents a new clustering genetic algorithm for data mining applications. A simple encoding scheme that yields to constant-length chromosomes is used, thus allowing the application of the standard genetic operators. Besides, a consistent algorithm, which avoids the problems of redundant codification and context insensitivity, is developed. In addition, a very simple heuristic is applied in order to generate the initial population. The individual fitness is determined based on the Euclidean distances among the objects, as well as on the number of objects belonging to each cluster. The clustering genetic algorithm is evaluated in the database known as Australian Credit Approval. The algorithm performs well considering that the rule set should provide low complexity measures (at most five clusters). The experimental results show that the proposed method is very promising.

1 Introduction

A primary goal of data mining in practice is *description*. Description focuses on finding human-interpretable patterns describing the data [1]. One of the description techniques used in data mining is clustering, which is a task where one seeks to identify a finite set of categories or clusters to describe the data [2]. A variety of clustering algorithms can be found in the literature [3,4,5,6] and clustering techniques have been used in a wide range of disciplines [7]. Some recent approaches optimize the clusters by means of genetic algorithms [8,9,10]. This paper describes a new clustering genetic algorithm and the results of its application to the problem of determining whether a person should receive credit or not.

A generic description of the clustering objective is to maximize homogeneity within each cluster while maximizing heterogeneity among clusters [7]. Basically, there are three kinds of clustering techniques: overlapping, hierarchical and partitioning. This work deals with the partitioning approach, which assigns each object to exactly one cluster. Thus, this work considers that clustering involves the partitioning of a set X of objects into a collection of mutually disjoint subsets C_i of X . Formally, let us consider a set of N objects $X = \{X_1, X_2, \dots, X_N\}$ to be clustered, where each $X_i \in \mathbb{R}^p$ is an attribute vector consisting of p real measurements. The objects must be clustered into non-overlapping groups $C = \{C_1, C_2, \dots, C_k\}$ where k is the number of clusters, such that [9]:

$$C_1 \cup C_2 \cup \dots \cup C_k = X, \quad C_i \neq \emptyset, \quad \text{and} \quad C_i \cap C_j = \emptyset \text{ for } i \neq j.$$

The problem with finding an optimal solution to the partition of N data into k classes is NP-complete [8]. Genetic algorithms are widely believed to be effective on NP-complete global optimization problems and they can provide good sub-optimal solutions in reasonable time [10]. Therefore, the application of genetic algorithms in clustering optimization problems can be very effective and efficient. The developed Clustering Genetic Algorithm (CGA) provides a set of rules in the following form:

$$\text{If } \{ v_{\min}^1 \leq a_1 \leq v_{\max}^1 \text{ and } \dots \text{ and } v_{\min}^p \leq a_p \leq v_{\max}^p \} \text{ then class } Y_i;$$

where a_i s are the attributes and the v_i 's are the corresponding maximum or minimum values from the obtained clusters for each class Y_i ($i = 1, 2, \dots, k$). These rules are obtained separately for each class. It means that the CGA searches for sub-classes that describe each class.

2 Clustering genetic algorithm

There are several complications concerning the application of genetic algorithms for clustering problems. The main one is to use a suitable *individual representation*. We use an encoding scheme that, when applied with the developed consistent algorithm, avoids the problems of redundant codification and context insensitivity [8].

2.1 Individual Representation

Considering that there are N objects to be clustered, a phenotype is represented as a one dimensional integer array with N elements. Considering that each data unit can be numbered from 1 to N , the i th element (i th gene) of a genotype represents the i th data unit. Considering that a low number of clusters provide more comprehensive rules, i.e. with lower complexity measures, we limited it to five for each class. Therefore, each gene of an individual has a value over the alphabet $\{1, 2, 3, 4, 5\}$, that is, the maximum number of clusters is equal to five.



Besides, the number of clusters is not given beforehand, but it is optimized by the genetic algorithm. For example, considering a dataset with 20 objects one can get the following cluster:

Genotype1: 22345123453321454552

This means that the cluster whose label is 2 is formed by 5 objects, corresponding to the set {1,2,7,13,20}, and the cluster whose label is 1 has 2 objects (6 and 14). The problem is that the straightforward encoding scheme above is highly redundant (there can be $k!$ chromosomes encoding the same solution to the original problem), i.e. the encoding is *one-to-many*. Considering the chromosome above, one can find more genotypes which also correspond to the same phenotypes like, for example: *Genotype2*=33451234514432515113; *Genotype3*=44512345125543121224; *Genotype4*=55123451231154232335; etc.

Therefore, the size of the space the CGA has to search is much larger than the original space of solutions. Indeed, the fitness function of the clustering problem depends exclusively on the grouping of the objects, rather than on the group label, that is, what matters is not the cluster label but the objects it contains. In addition, there is another problem caused by this encoding scheme. It is the undesirable effect of casting context-dependent information *out of context* under the standard crossover, i.e. the context insensitivity [8]. For example, if *genotype2* is crossed over with *genotype3*, the resulting genotype is supposed to be equal to their parents', which represent the same phenotype. But this rarely happens. In order to make it more easily observed, let us apply the two point crossover to the following genotypes:

Genotype6: 2 | 2222 | 111113333344444
Genotype7: 4 | 4444 | 33333555511111

Then resulting:

Child1: 24444111113333344444
Child2: 4222233333555511111

One can observe that the children are different from their parents, that is, they represent different phenotypes. On top of that, it is verified that the second child represents five clusters, whereas their parents represent just four. These problems are solved by the developed Consistent Algorithm (CA). This algorithm allows the *reorganization* of the genotypes. Lets consider a genotype G - where $G[i]$ represents the cluster label for the "i" gene - and an auxiliary N -length string S . Then, the CA is as depicted in Figure 1.

Applying this algorithm, all the genotypes {1,2,3,4} above will result in the same one: 11234512342215343441. Therefore, the CA allows the CGA to search in space of a one gene for one cluster basis, i.e. there is only one genotype for each different phenotype. The Consistent Algorithm (CA) is an important

part of the developed Clustering Genetic Algorithm (CGA) that can be basically described as follows:

- 1 – Initialize a population of chromosomes;
- 2 – Apply the Consistent Algorithm to the current population;
- 3 – Evaluate each chromosome in the population and apply a linear normalization;
- 4 – Select individuals by proportional selection;
- 5 – Apply crossover and mutation to the selected individuals;
- 6 – Delete chromosomes and insert the new ones that will form the current population;
- 7 – If the convergence is attained, stop; if not, go to step 2.

```

1 – Make all N-genes of S equal zero;
2 – Label = 1;
3 – For i = 1,...,N do:
    If S[i]=0 then:
        Aux = G[i];
        For j = 1,...,N do:
            If (G[j]=aux and S[j]=0) then:
                G[j]=Label;
                S[j]=1;
            End if;
        End For;
        Label=Label+1;
    End If;
End For.

```

Figure 1: Consistent Algorithm.

2.2 Fitness Function

The fitness function must reflect the way each individual is, in comparison to the others, becoming closer to the optimal solution. The fitness function has to improve as the size of the cluster grows. Of course a group formed by only one object has little meaning in clustering problems. Besides, it is necessary to avoid the trivial solution, which consists in putting all the objects into the same group. In this study, the individual fitness is determined based on the euclidean distances among the objects as well as on the number of objects belonging to each cluster. Basically, the fitness function can be settled as follows:

$$F(\text{dex}, \text{din}, \text{factor}) = \text{dex} - \text{din} + \text{factor}. \quad (1)$$

Where *dex* is the average *external distance*, *din* is the average *internal distance* of the clusters represented by a chromosome, and *factor* is a function of the number of objects clustered by each group. Thus, the more *dex* grows and



din decreases, the higher the value of the fitness function, what implies in better solutions to the clustering problem. More specifically, *dex* is calculated for each chromosome as follows:

- 1 – Compute the centroids of each cluster represented by the chromosome;
- 2 – Calculate the Euclidean distances among all these centroids;
- 3 – Sum up all the distances computed in step 2);
- 4 – Divide this value by the number of cluster combinations, i.e. by

$$C_k^2 = k! / [2! (k-2)!].$$

One can observe that k is the number of clusters represented by the chromosome and can vary from one to five. The average internal distance *din* is calculated for each chromosome as follows:

- 1 – Compute the centroids of each cluster represented by the chromosome;
- 2 – For each cluster, calculate the Euclidean distances among its centroid and the objects belonging to it. Sum up all these distances and divide this result by the number of objects belonging to the cluster in consideration;
- 3 – Sum up all the distances computed in step 2), which are the average internal distances to each cluster and divide this value by the number of clusters.

It is believed that all the terms of equation (1) should have approximately the same impact on the fitness function value. The main concern is about the term *factor*. As it depends basically on the number of elements belonging to each cluster, it should not be, on average, higher than the other ones. If that happens, the algorithm will search for clusters formed by a lot of elements, regardless of their similarity. Besides, as the maximum number of clusters is set to five, it is desirable to have, in principle, the patterns equally distributed in the five clusters related to the final solution.

Therefore, considering that n_t is the total number of activation patterns and $n_{\max}$ is the number of elements of the biggest cluster, one can impose the condition of having equal-size clusters by doing *Factor* be a function of $(n_t - n_{\max})$. Thus, considering that the final solution should have five equal-size clusters, $n_{\max}$ should be equal to $(n_t/5)$. Besides, *Factor* should have the same influence as the other ones in eqn.(1). Thus, one can multiply the term *Factor* by a constant K in order to favor this situation, such that:

$$\text{Factor} = (n_t - n_t/5) \cdot K. \quad (2)$$

Doing so, one has to find a suitable value for the constant K . One way of having some control over the influence of K in eqn.(2) is by making *factor* be, on average, approximately equal to the average distance among all the objects (d_m). Thus, if one considers that the best solution has five clusters with an equal number of elements, *Factor* is calculated as follows:

$$\text{Factor} = 5(n_t - n_{\max}) \cdot d_m / (4n_t). \quad (3)$$

The employed methodology tends to favor big clusters (containing many objects) with low internal distances and high external distances. In order to provide a mutual comparison among the chromosomes, a linear normalization [11,12] is applied to the fitness function values. First, the fitness function values are ranked. Then, they are replaced by:

$$F' = R(i) \cdot [F'(\max) - F'(\min)] / \text{popsize}. \quad (4)$$

Where $R(i)$ is the position of the i chromosome in the ranking, $F'(\max)$ is the highest fitness function value, $F'(\min)$ is the lowest fitness function value, and popsize is the number of individuals in the population.

2.3 Crossover and Mutation

The individuals that make part of each generation are selected accordingly to the well known roulette wheel selection strategy [11,12]. Besides, an elitist strategy is used, that is, the best individual of each generation is always copied into the succeeding generation as well as 20% of the best individuals are always selected for crossover and mutation. All the individuals selected are crossed over. The first chromosome is crossed over with the last one, the second chromosome with the penultimate one and so on. A two-point crossover technique is used due to its suitability for grouping problems [8].

All the genes of an individual have a chance of being mutated. A seed s is computed for each gene and whenever it is smaller than the mutation probability p_m , one gene of that chromosome is likely to be mutated. It happens because whether $s \leq p_m$ a new seed s_m is computed for establishing the allele value among the five possible ones. Considering that mutation is too destructive once the CGA begins to reach a good solution of the clustering problem [8], p_m is equal to a typically low value, that is 0.004. Thus, each gene has a 0.32% chance of being mutated. The best genotype of each generation is always stored.

2.4 Initial Population

Considering clustering problems, the generation of random initial populations often leads to negative fitness function values, which are undesirable when using the roulette wheel selection strategy. Therefore, a very simple heuristic is applied in order to generate an initial population whose individuals do not have negative fitness function values.

The attribute values for each example of the database are computed and converted to standard scores (also known as Z scores) by subtracting the mean and dividing it by the standard deviation of each attribute. Thus, it is possible to eliminate the bias introduced by the differences in the scales of the several attributes, i.e. the mean of each attribute is equal to zero, whereas the standard deviation is equal to one.

The application of standard scores allows the utilization of a very simple heuristic to form the initial population. First, all the distances among the objects and the centroid of each class are calculated. Then, it is possible to know the highest distance value, which will be here called M . Let us consider that the distance between the object i and the centroid is called $d[i]$, and that N is the number of objects. We establish four initial solutions and the other ones are formed by means of mutations of these solutions. The four *initial* genotypes are formed as expressed according to Figure 2. The other genotypes are formed by means of mutations of these ones, considering that there is a 30% chance of each gene to be mutated.

Genotype 1: For $i=1,...,N$ do: If $d[i] < 0.1M$ then $G[i]=1$; If $0.1M \leq d[i] < 0.2M$ then $G[i]=2$; If $0.2M \leq d[i] < 0.3M$ then $G[i]=3$; If $0.3M \leq d[i] < 0.4M$ then $G[i]=4$; If $d[i] \geq 0.4M$ then $G[i]=5$; End For.	Genotype 2: For $i=1,...,N$ do: If $d[i] < 0.2M$ then $G[i]=1$; If $0.2M \leq d[i] < 0.3M$ then $G[i]=2$; If $0.3M \leq d[i] < 0.4M$ then $G[i]=3$; If $0.4M \leq d[i] < 0.5M$ then $G[i]=4$; If $d[i] \geq 0.5M$ then $G[i]=5$; End for.
Genotype 3: For $i=1,...,N$ do: If $d[i] < 0.2M$ then $G[i]=1$; If $0.2M \leq d[i] < 0.4M$ then $G[i]=2$; If $0.4M \leq d[i] < 0.6M$ then $G[i]=3$; If $0.6M \leq d[i] < 0.8M$ then $G[i]=4$; If $d[i] \geq 0.8M$ then $G[i]=5$; End for.	Genotype 4: For $i=1,...,N$ do: If $d[i] < 0.3M$ then $G[i]=1$; If $0.3M \leq d[i] < 0.4M$ then $G[i]=2$; If $0.4M \leq d[i] < 0.5M$ then $G[i]=3$; If $0.5M \leq d[i] < 0.6M$ then $G[i]=4$; If $d[i] \geq 0.6M$ then $G[i]=5$; End for.

Figure 2: Generating the Initial Population.

3 Case studied – Australian credit approval database

The clustering genetic algorithm was tested on the *Australian Credit Approval Database*, which is a known benchmark for data mining applications. This database concerns about credit card applications and is available at www.ncc.up.pt/liacc/ML/statlog/datasets.html. The database contains 690 instances - 307 of people who were granted credit (class 1) and 383 of people who were not granted credit (class 2) - formed by 14 attributes A_i $\{i = 1, \dots, 14\}$ and the class label. There are six numerical attributes and 8 categorical ones. However, all the attribute names and values have been changed to meaningless symbols in order to protect the confidentiality of the data.

For this example, one employs a population formed by 200 individuals, and the fitness function is given by expression (1). The CGA is stopped after 2,000 generations. Considering *class 1*, the CGA converged after 1,106 generations and the fitness function value evolved from 2.64 to 4.55. The obtained genotype

represents five clusters – *C1*, *C2*, *C3*, *C4* and *C5* - whose number of objects are equal to 156, 140, 9, 1 and 1 respectively.

Considering *class 2*, the CGA converged after 1,930 generations and the fitness function value evolved from 15.31 to 33.46. The obtained genotype represents five clusters – *C6*, *C7*, *C8*, *C9* and *C10* - whose number of objects are equal to 77, 77, 76, 77 and 76 respectively. The average CPU time for *class 1* was equal to two hours and 2 minutes, and for *class 2* it was equal to two hours and 28 minutes (Pentium 233Mhz, 32 RAM).

The clusters formed by only one object – *C4* and *C5* - were considered inadequate for the knowledge acquisition process, because they can represent noisy data. However, this strategy should be validated by a domain expert. Besides, the other clusters are evaluated considering two aspects. On the one hand, the cluster has to represent the maximum number of examples of its own class. On the other hand, the cluster has to represent the minimum number of examples of the opposite class.

Therefore, the following strategy to rank the clusters has been employed: in case a particular cluster represents an instance of its class, it receives one point as well as in case it does not represent an instance of the opposite class. Thus, the number of instances that a cluster represents, considering its own class, is computed. This result is summed up to the number of instances of the opposite class that this cluster does not represent. These results are described in Table 1, whereas Table 2 shows the best results in terms of the average classification rate.

Table 1. Cluster Classification.

Cluster	Class 1	Class 2	Total	Ranking
C1	300	5	305	3 (Class 1)
C2	228	328	556	1 (Class 1)
C3	94	256	350	2 (Class 1)
C6	161	316	477	3 (Class 2)
C7	228	270	498	1 (Class 2)
C8	239	209	448	4 (Class 2)
C9	156	287	443	5 (Class 2)
C10	189	301	490	2 (Class 2)

One observes that the number of objects belonging to each cluster listed on Table 1 is different from that obtained from the CGA. It happens because this table considers the knowledge structure represented by each cluster. Therefore, if an object has its variable values belonging to the variable range of values of a cluster, it makes part of this cluster. Besides, it is possible to see, by means of Table 2, that the best solution is represented by using the best cluster of *class 1* and the two best clusters of *class 2*.

Table 2. Average Classification Rate.

Class 1	Class 2	Classification
3 best clusters	5 best clusters	13 %
2 best clusters	4 best clusters	45 %
Best cluster	3 best clusters	57 %
Best cluster	2 best clusters	61 %
Best cluster	Best cluster	52 %
3 best clusters	-----	45 %
-----	5 best clusters	69 %
Best cluster	-----	81 %
-----	Best cluster	72 %

Obviously it is possible to get better results considering only one class at a time. For example, one can consider that the best knowledge structure is formed by the best cluster of *class 1*, so that the average classification rate is equal to 81%. Although this average classification rate is the best one, the rules will only describe *class 1*, whereas *class 2* will be considered as the *default* one. This strategy does not allow to know the characteristics of both classes. Therefore, one recommends to consider both classes for the knowledge representation task. Thus, the best solution can be translated to the set of rules expressed in Figure 3. This rule set provides an average classification rate equals to 61%.

If { $0 \leq A_1 \leq 1$ and $16 \leq A_2 \leq 64$ and $0 \leq A_3 \leq 15$ and $1 \leq A_4 \leq 2$
and $1 \leq A_5 \leq 14$ and $1 \leq A_6 \leq 8$ and $0 \leq A_7 \leq 13$ and $A_8 = 1$
and $0 \leq A_9 \leq 1$ and $0 \leq A_{10} \leq 23$ and $0 \leq A_{11} \leq 1$ and $A_{12} = 2$
and $0 \leq A_{13} \leq 470$ and $1 \leq A_{14} \leq 15,109$ } then *class 1*;
If { $0 \leq A_1 \leq 1$ and $17 \leq A_2 \leq 75$ and $0 \leq A_3 \leq 26$ and $1 \leq A_4 \leq 2$
and $1 \leq A_5 \leq 9$ and $1 \leq A_6 \leq 8$ and $0 \leq A_7 \leq 7$ and $0 \leq A_8 \leq 1$
and $0 \leq A_9 \leq 1$ and $0 \leq A_{10} \leq 4$ and $0 \leq A_{11} \leq 1$ and $1 \leq A_{12} \leq 2$
and $0 \leq A_{13} \leq 550$ and $1 \leq A_{14} \leq 1,001$ } then *class 2*;
If { $0 \leq A_1 \leq 1$ and $16 \leq A_2 \leq 57$ and $0 \leq A_3 \leq 13$ and $1 \leq A_4 \leq 2$
and $1 \leq A_5 \leq 14$ and $1 \leq A_6 \leq 8$ and $0 \leq A_7 \leq 3$ and $0 \leq A_8 \leq 1$
and $0 \leq A_9 \leq 1$ and $0 \leq A_{10} \leq 5$ and $0 \leq A_{11} \leq 1$ and $1 \leq A_{12} \leq 2$
and $0 \leq A_{13} \leq 2,000$ and $1 \leq A_{14} \leq 4,501$ } then *class 2*;

Figure 3: Knowledge Structure.

4 Conclusions

In this paper we have presented a new clustering genetic algorithm. The developed *Consistent Algorithm* allows the utilization of the standard genetic operators and also avoids the problems of redundant codification and context insensitivity, which are serious problems to the use of genetic algorithms in



clustering problems. A very simple heuristic is applied to generate the initial population. The developed clustering genetic algorithm is evaluated in the database known as *Australian Credit Approval* and the results of this application are here described. On the one hand, the results show that the method is very promising, mainly considering that one attempts to establish a complexity limit for the knowledge structure. On the other hand, it is possible to get better results by applying a better heuristic to set up the initial population. Besides, it is possible to find a better fitness function, as well as it is necessary to find the best parameters for the clustering genetic algorithm.

5 References

- [1] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., From Data Mining to Knowledge Discovery (Chapter 1). *Advances in Knowledge Discovery and Data Mining*, ed. Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: AAAI Press, pp. 1-36, 1998.
- [2] Titterton, D. M., Smith, A. F. M., Makov, U. E., *Statistical Analysis of Finite Mixture Distributions*. Chichester, U.K.: John Wiley and Sons, 1985.
- [3] Anderberg, M. R., *Cluster Analysis for Applications*, Academic Press, 1973.
- [4] Hartigan, J. A., *Clustering Algorithms*, John Wiley & Sons, 1975.
- [5] Kaufman, L., Rousseeuw, P. J., *Finding Groups in Data – An Introduction to Cluster Analysis*, Wiley Series in Probability and Mathematical Statistics, 1990.
- [6] Arabie, P., Hubert, L. J., DeSoete, G., *Clustering and Classification*, ed. P. Arabie, L.J. Hubert, G. DeSoete, World Scientific, 1999.
- [7] Arabie, P., Hubert, L. J., An Overview of Combinatorial Data Analysis (Chapter 1). *Clustering and Classification*, ed. P. Arabie, L.J. Hubert, G. DeSoete, World Scientific, 1999.
- [8] Falkenauer, E., *Genetic Algorithms and Grouping Problems*, John Wiley & Sons, 1998.
- [9] Cole, R. M., *Clustering with Genetic Algorithms*, Master of Science Thesis, Department of Computer Science, University of Western Australia, 1998.
- [10] Park, Y., Song, M., A Genetic Algorithm for Clustering Problems, *Proc. of the Genetic Programming*, University of Wisconsin in Madison, pp. 568-575, 1998.
- [11] Davis, L. *Handbook of Genetic Algorithms*, International Thompson Computer Press, 1996.
- [12] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley Longman, Inc., 1989.