Natural Coding: A More Efficient Representation for Evolutionary Learning*

Raúl Giráldez, Jesús S. Aguilar-Ruiz, and José C. Riquelme

Department of Computer Science, University of Seville
Avenida Reina Mercedes s/n, 41012 Sevilla, Spain
{giraldez,aguilar,riquelme}@lsi.us.es

Abstract. To select an adequate coding is one of the main problems in applications based on Evolutionary Algorithms. Many codings have been proposed to represent the search space for obtaining decision rules. A suitable representation of the individuals of the genetic population can reduce the search space, so that the learning process is accelerated by decreasing the number of necessary generations to complete the task. In this sense, natural coding achieves such reduction and improves the results obtained by other codings. This paper justifies the use of natural coding by comparing it with hybrid coding that joins well-known binary and real representations. We have tested both codings on a heterogeneous subset of databases from the UCI Machine Learning Repository. The experiments' results show that natural coding improves the quality of the obtained knowledge-model using only one third of the generations that hybrid coding needs as well as a smaller population.

Keywords: Evolutionary Algorithms, Coding, Supervised Learning

1 Introduction

Machine Learning is used when we want to build a knowledge model from a training dataset and predict the outcome of a new unseen instance. Normally, the instances of the dataset are named examples and each example is formed by attributes. These attributes can be classified by different criteria according to the kind of values that they take. The most common classification is in continuous and discrete attributes. If an attribute takes values in a real and infinite domain, we say that it is continuous. On the contrary, if the domain is a finite set of values, we name it as a discrete attribute. Usually, one of these attributes is selected as a decision attribute or class. Thus, the knowledge model is built to predict the value that the class takes in new examples depending on the other attributes values. When the class of the training data is known (labeled dataset), we work in the Supervised Learning field.

* This research was supported by the Spanish Research Agency CICYT under grant TIC2001-1143-C03-02.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 979–990, 2003.

The knowledge model can be represented by different structures: decision rules, association rules, decision trees, etc. In Supervised Learning, the structures generated are usually decision rules or trees. The databases have a set of attributes or characteristics that define each example. A decision rule establishes conditions which the examples must fulfil in order to be classified by this rule. These conditions affect the values that the attributes can take in that if the attributes of an example meet all the conditions that a rule establishes then we say that the rule covers the example, independent of whether it classifies such example correctly or not. The usual representation of rules generated by learning systems is shown in Figure 1.

If $Cond_1$ **and** $Cond_2$ **and**... **and** $Cond_M$ **then** $Class = C$

Fig. 1. Decision Rule.

where $Cond_i$ is the condition that the i^{th} attribute of an example has to satisfy in order to be classified with class C, and M is the number of attributes in the database. Logically, the case can arise where an attribute does not appear in the rule, from which we assume that the condition concerning the attribute is always evaluated as *true*. When an attribute (a_i) is continuous, the condition ($Cond_i$) takes the form $a_i \in [l_i, u_i]$, restricting the range of values of the attribute to the interval defined by the lower (l_i) and the upper (u_i) bound. On the other hand, when the attribute is discrete, the condition takes the form $a_i \in \{v_1, v_2, \ldots, v_k\}$, where the values $\{v_1, v_2, \ldots, v_k\}$ are not necessarily all those the attribute can take.

In literature, there are many methods that acquire the inherent knowledge from a labeled dataset and generate a structure that represents it: CN2 [6,7], AQ-based systems [13],OC1 [15], C4.5 [16], SEE5.0 [17], SIA [20], among others. This work is focused on those methods that apply Evolutionary Algorithms (henceforth EA) in the training phase, and specifically in the tool HIDER (Hierarchical Decision Rules) [2]. This tool generates a set of hierarchical decision rules by applying an EA whose population is a set of encoded rules. Thus, each individual is a decision rule where each attribute is coded by one or several genes. Different versions of HIDER have used different codings for the individuals of population: binary, real and hybrid. This work is focused on a new version of this tool, called HIDER*, which applies the natural coding [1] to represent the individuals. Thus, we compare the two last versions, HIDER and HIDER*, that use the hybrid and natural coding respectively. Both codings can treat continuous and discrete attributes. Hybrid coding applies binary coding for discrete attributes and real coding for continuous. Natural is an original coding that represents both kinds of attributes by means of a simple natural number. As it is explained later, the main advantage of natural coding is the reduction of the search space.

Our goal in this paper is to present the advantages of natural coding against hybrid coding. HIDER and HIDER* were run and their performances measured. The results show that HIDER* finds solutions before HIDER, since natural coding

reduces the number of candidate solutions. Thus, HIDER* can be run with fewer generations and with a smaller population size than HIDER.

The rest of this paper is organized into the following: Section 2 describes the EA used by HIDER and HIDER*. The characteristics of hybrid coding and natural coding are presented in Sections 3 and 4 respectively. The experimental results of our research are presented in Section 5. Finally, in Section 6, the conclusions of this work are summarized.

2 Algorithm

HIDER is a tool that produces a hierarchical set of rules. When a new example needs to be classified, the set of rules is sequentially evaluated according to its hierarchy, so if the example does not fulfil a rule, the next one in the hierarchy order is evaluated. This process is repeated until the example matches every condition of a rule and is classified with the class that such rule establishes.

HIDER uses an EA to search for the best solutions. Since the aim is to obtain a set of decision rules, the population of the EA is formed by some possible solutions. Each genetic individual is a rule that evolves by applying the mutation and crossover operators. In each generation, some individuals are selected according to their goodness of fit and they are included in the next population along with their offspring. Thus, the performance of the tool is influenced by two factors: the fitness function and the coding. The fitness function used in this research is

$$f(r) = N - CE(r) + G(r) + coverage(r) \tag{1}$$

where r is an individual; N is the number of examples being processed; $CE(r)$ is the class error, i.e. the number of examples belonging to the region defined by the rule r but they do not have the same class; $G(r)$ is the number of examples correctly classified by r; and $coverage(r)$ gives the proportion of the search space covered by the rule. This fitness function is described in detail in [2]. The goal of our work is focused on the coding, so the another factor is not analyzed in detail. In fact, HIDER and HIDER* use the same EA and the same fitness function. The difference between both is in the coding. HIDER applies hybrid coding, whereas HIDER* uses natural coding.

The pseudocode of HIDER is shown in Figure 2. The main algorithm is a typical sequential covering method [14], where the algorithmic function that produces the rules is an EA. Each call for this function (line 8) generates only one rule that is inserted into the final set of rules (line 9) and is used to eliminate examples from the training data (line 10). The evolutionary function is started again with the reduced training data. This loop is repeated until the set of training data is empty.

The function *EvoAlg* has a set of examples as an input parameter. It returns a rule which is the best individual of the last generation. The initial population is built randomly by the function *InitializePopulation*. Some examples are randomly selected and individuals that cover such examples are generated. After

```
1 Procedure HIDER
2  Input: T: File of examples (Training file)
3  Output: R: Set of rules (Sorted set)
4  begin
5    R := ∅
6    initialSize := |T|
7    while |T| > initialSize
8      r:=EvoAlg(T)
9      R:=R ⊕ r
10     DeleteCoveredExamples(T,r)
11   end while
12 end HIDER
13 Function EvoAlg (T: File of encoded-examples) ret(r: Rule)
14  begin
15    InitializePopulation(P)
16    For i:=1 to num_generations
17      Evaluate(P)
18      next_P:=SelectTheBestOf(P)
19      next_P:=next_P+Replicate(P)
20      next_P:=next_P+Recombine(P)
21      P:=next_P
22    end for
23    Evaluate(P)
24    return SelectTheBestOf(P)
25 end EvoAlg
```

Fig. 2. Pseudocode of HIDER.

initializing the population, the for-loop repeats the evolutionary process a number of times which is determined by the parameter *num_generations*. In each iteration, the individuals of the population are evaluated according to a defined fitness function, thus each individual acquires a "goodness" (function *Evaluate*). The best individual of every generation is replicated to the next one (elitism). Later, a set of individuals are selected through the roulette wheel method and replicated to the next generation. Finally, another set of individuals are recombined and the offspring is included in the next generation. The selection of these individuals is also carried out by means of the roulette wheel. Once the loop finishes, the best individual of the last generation is returned.

3 Hybrid Coding

After studying other EA-based approaches proposed in the bibliography, we deduced that a combination of binary coding and real coding would be suitable for our problem. First, we analyzed two well-known systems: GABIL [8] and GIL [11]. These concept learners use binary coding, assigning a bit to each value of an attribute. A value of one, in a bit, implicates that the value is present, so that several bits could be active for the same attribute. This implies that binary coding is a good election for encoding symbolic attributes. Nevertheless, it is not suitable for continuous attributes, because the size of their alphabet is very large (theoretically infinite) and this aspect does not allow a complete search. Non-binary codings have been widely researched in literature [3,4,12,

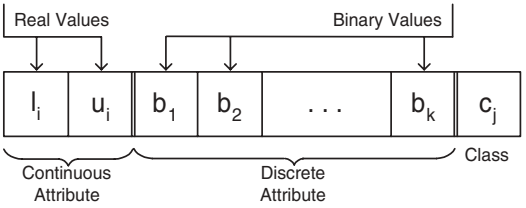

Fig. 3. An hybrid individual with a continuous attribute and a discrete one.

21,22]. These jobs show that real coding is more appropriate than binary in continuous domains.

Hybrid coding tries to profit from the advantages of binary and real coding. Thus, it mixes both codings by using the real to represent the boundaries of the intervals in continuous domains, and the binary to encode the discrete attributes. Figure 3 explains the representation for continuous and discrete attributes in hybrid coding. The continuous attribute is encoded by two real values being the lower (l_i) and the upper bound (u_i) of the interval that the rule establishes for such attribute. The discrete attribute is represented by a number k of bits, where each of them denotes the presence or absence of a discrete value in the condition of the rule. The last value is the class, which is not usually encoded. Each class is represented as a integer, so that for a number n of classes, the value c_j will belong to the set {0, 1, 2, ..., n-1}.

With regard to the genetic operators, crossover and mutation for hybrid coding are described in detail in [2]. The crossover for continuous attributes is an extension of Radcliffe's [18] to parents coded as intervals. If two parents, a and b, establish the intervals $[l_i^a, u_i^a]$ and $[l_i^b, u_i^b]$ respectively for the attribute a_i, then the interval of the offspring ($[l_i^o, u_i^o]$) fulfils expressions in Equation 2. In case of discrete attributes, the uniform crossover is carried out [19].

$$\begin{array}{c} l_i^o \in [min(l_i^a, l_i^b), max(l_i^a, l_i^b)] \\ u_i^o \in [min(u_i^a, u_i^b), max(u_i^a, u_i^b)] \end{array} \tag{2}$$

In order to apply the mutation to a continuous attribute, a gene (g_i) is randomly selected. Such gene can be the lower bound (l_i) or the upper (u_i) bound of the interval. The mutation consist in replacing the gene g_i with $g_i \pm \delta$, where δ is the smaller HOEM (Heterogeneous Overlap-Euclidean Metric [23]) between two examples of the training dataset. Thus, an interval is expanded ($l_i - \delta$ or $u_i + \delta$) or contracted ($l_i + \delta$ or $u_i - \delta$). When the attribute is discrete, the mutation changes the gene (a bit) depending on the discrete-value mutation probability, so that some values are included or excluded from the condition.

4 Natural Coding

The ideal coding must fulfil the following properties: completeness, coherence, uniformity (uniqueness), simplicity, locality, consistency and minimality. To find

Table 1. Coding for a continuous attribute.

Cutpoints	2.5	3.9	4.7	5.0	6.2
1.4	**1** ≡ [1.4, 2.5]	**2** ≡ [1.4, 3.9]	**3** ≡ [1.4, 4.7]	**4** ≡ [1.4, 5.0]	**5** ≡ [1.4, 6.2]
2.5	–	**7** ≡ [2.5, 3.9]	**8** ≡ [2.5, 4.7]	**9** ≡ [2.5, 5.0]	**10** ≡ [2.5, 6.2]
3.9	–	–	**13** ≡ [3.9, 4.7]	**14** ≡ [3.9, 5.0]	**15** ≡ [3.9, 6.2]
4.7	–	–	–	**19** ≡ [4.7, 5.0]	**20** ≡ [4.7, 6.2]
5.0	–	–	–	–	**25** ≡ [5.0, 6.2]

a suitable representation is very difficult but to find one that fulfils every aforementioned property is practically impossible. Thus, we try to design a coding that keeps at least two well-known principles proposed by Goldberg in [10]: the principle of meaningful building blocks and the principle of minimal alphabets.

After analyzing the problem, we developed a coding, named "natural" [1], which only uses natural numbers to represent the set of values that can take part of the decision rules, independently of the kind of attributes (continuous or discrete). In particular, natural coding encode each condition of a rule with only one gene that is a natural number. Thus, this coding fulfils two aforementioned properties: uniqueness (every individual has an unique representation) and minimality (the length of coding must be as short as possible). Although continuous and discrete values are encoded as natural numbers, the meaning of this number is different in both cases.

As in regards to continuous attributes, a method that calculates the cutpoints in range of values for each continuous attribute is applied. The cutpoints are the possible bounds of intervals that a condition can set. In principle, these cutpoints can all be of values the attribute takes in the training dataset. However, this choice generates too many cutpoints. In order to reduce the number of cutpoints, we apply the method named USD [9]. This method produces a set of bounds that maximize the goodness of the possible intervals. Such an aspect has a favorable influence on the rules that HIDER* is going to obtain later. Once the cutpoints are calculated, a natural number is assigned to each possible combination of bounds, so every possible interval is represented by a natural number. Table 1 shows an example of coding for a continuous attribute where the set of cutpoints generated by USD is {1.4, 2.5, 3.9, 4.7, 5.0, 6.2}.

In general, if k is the number of cutpoints, the rows are labeled from the first one to $(k-1)^{th}$ and the column are labeled from the second one to k^{th} cutpoint. Thus, each element of the table is a natural number that encodes the interval defined by its row and column.

In case the attributes are discrete, the natural coding is obtained from a binary coding similar to that used in GABIL and GIL. In decision rules, a condition can establish a set of discrete values that the attribute must take to classify an example. Each gene that represents a condition for a discrete attribute is also a natural number. The binary representation of this gene is a number where there is a bit for each different discrete value that an attribute can take. Thus, if Ω is the set of possible values for a discrete attribute, then the natural number that encodes the corresponding condition of the rule is in interval $[0, 2^{|\Omega|} - 1]$. If a value is included in the condition, its corresponding bit

Table 2. Coding for a discrete attribute.

Discrete values					Natural
white	red	green	blue	black	Coding
0	0	0	0	0	–
0	0	0	0	1	**1**
0	0	0	1	0	**2**
0	0	0	1	1	**3**
⋮	⋮	⋮	⋮	⋮	⋮
0	1	0	1	1	**11**
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	0	**30**
1	1	1	1	1	**31**

is equal to 1, otherwise it is 0. The natural coding for this gene is the conversion of the binary number into a natural one. Table 2 shows an example for a discrete attribute with five different values: *white*, *red*, *green*, *blue* and *black*.

The mutation and crossover operators are described in detail in [1]. For continuous attributes, mutation is a shift (up, down, left or right) in the coding table (see Table 1). Such shift mutates the gene changing the bounds of the interval. The crossover consists in calculating the intersection of the rows and columns of both parents, so that the obtained set of natural numbers is the offspring. For discrete attributes, the mutation is a transformation of the value that encodes the gene, by changing some bits of the binary code associated with the natural number. The crossover is based on the mutation. Each gene that take part in the crossover provides a set of candidates obtained from its possible mutations joined to itself. The offspring of two genes is a random selection of numbers from the intersection of both sets of candidates. Although the natural operators seem to be very complex processes, in fact, they are very simple algebraic operations. They do not imply any conversion between binary and natural numbers and do not need any operation with arrays. As shown in [1], the natural operators are algebraic operations with a low computational cost.

Figure 4 illustrates the differences between natural and hybrid coding. It shows two individuals that encode the same rule. The attribute AT_1 is continuous whereas AT_2 is discrete. Both attributes are encoded according to Tables 1 and 2.

We can see that natural coding is simpler, since the hybrid needs eight genes to code the rule whereas the natural encodes it with only three genes. Natural coding minimizes the size of individuals, assigning only one gene for each attribute. Hybrid uses two genes for continuous attributes and —Ω— for discrete, where —Ω— is the number of different values that an attribute can take.

5 Results

In order to show the quality of natural coding, we applied HIDER and HIDER* to a set of databases from UCI Repository [5]. As we mentioned in Section 2, both tools use the same EA, although HIDER uses hybrid coding, to the contrary of

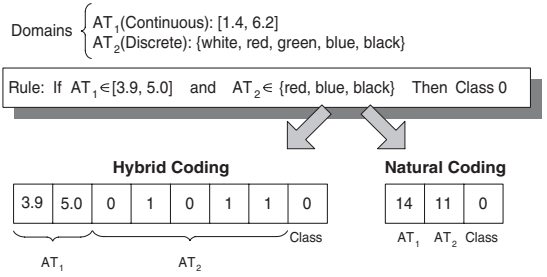

Fig. 4. Hybrid individual vs. Natural individual.

Table 3. Parameter of HIDER.

Parameter	HIDER (hybrid)	HIDER* (natural)
Population Size	100	70
Number of Generations	300	100
Replication	20%	20%
Recombination	80%	80%
Individual Mutation Probability	0.5	0.5
Gen Mutation Probability	$\frac{1}{\|attributes\|}$	$\frac{1}{\|attributes\|}$
Discrete-value Mutation Probability	$\frac{1}{\|values\|}$	$\frac{1}{\|values\|}$

HIDER* which uses natural coding. Both were run with the same crossover and mutation parameters, but with a different number of individuals and generations. Table 3 shows the parameters used in each case. The lower values that HIDER* needs for the population size and number of generations are due to the reduction of the search space by natural coding. Thus, HIDER* used only 100 generations and 70 individuals to obtain similar numbers to those obtained by HIDER, which needed 300 and 100 respectively (see Table 4).

The databases used in the experiments were 16, some of which contain only continuous attributes, others contain only discrete attributes and the remainder include both types of attributes. Thus, we can compare the behaviour of natural and hybrid coding with both types of attributes. To measure the performance of each method, a 10-fold cross-validation was achieved with each database. The values that represent the performance are the error rate (ER) and the number of rules (NR) obtained. The ER is the average number of misclassified examples expressed as a percentage and the NR is the average number of rules for the 10-fold cross-validation. The algorithms were run on the same training sets and the knowledge structures tested using the same test sets, so the results were comparable. With these conditions and the aforementioned parameters for the EAs, the results obtained by applying both codings are given in Table 4.

In Table 4, the first column shows the databases used in the experiments; the next two columns give the ER and NR obtained respectively by HIDER with hybrid coding for each database. Likewise, the fourth and fifth columns give the ER and NR for HIDER* with natural coding. The last two columns show a

Table 4. Comparing HIDER and HIDER*.

	HIDER (hybrid)		HIDER* (natural)		Improvement	
Database	ER	NR	ER	NR	ϵ_{er}	ϵ_{nr}
breast-c	4.3	2.6	4.06	2	1.06	1.3
bupa	35.7	11.3	37.35	4.2	0.96	2.69
cleve	20.5	7.9	25.33	5.9	0.81	1.34
german	29.1	13.3	27.4	8	1.06	1.66
glass	29.4	19	35.24	11.7	0.83	1.62
heart	22.3	9.2	21.85	4.3	1.02	2.14
hepatiti	19.4	4.5	16.67	3.7	1.16	1.22
horse-co	17.6	6	20	11.1	0.88	0.54
Iris	3.3	4.8	3.33	3.2	0.99	1.5
Lenses	25	6.5	25	4.5	1	1.44
mushroom	0.8	3.1	1.18	3.5	0.68	0.89
pima	25.9	16.6	25.66	5.1	1.01	3.25
vehicle	30.6	36.2	33.81	19.7	0.91	1.84
vote	6.4	4	4.42	2.2	1.45	1.82
wine	3.9	3.3	8.82	5.6	0.44	0.59
zoo	8	7.2	4	7.9	2	0.91
Average	17,64	9,72	18,38	6,41	1,02	1,55

measure of improvement for the error rate (ϵ_{er}) and the number of rules (ϵ_{nr}). The ϵ_{er} coefficient was calculated by dividing the error rate for HIDER by the corresponding error rate for HIDER*. The same operation was carried out to obtain ϵ_{nr}, but by using the number of rules for both tools. Finally, the last row shows the average results for each column. As we can observe, HIDER* does not attain a reduction in ER for 8 out of 16 datasets. Nevertheless, on average, it improves on HIDER, although this improvement is very small (2%). As regards to number of rules, the results are more significant, since HIDER* obtains a smaller number of rules in 12 out of 16 cases, with an average improvement of 55%.

Although the results show that HIDER* has a better performance, we must not forget that those numbers were obtained using a smaller number of generations and individuals of a genetic population. In particular, HIDER* needed one third of the generations and fewer than three quarters of the population size invested by HIDER. The reduction of the search space that natural coding gets, allows HIDER* to obtain these results with such parameters.

In Table 5, the length of individuals of the genetic population for the hybrid and natural coding are shown. The first column contains the databases. The second shows the number of continuous attributes (NC), if any, for each database. Likewise, the next column gives the number of discrete attributes (ND) along with the total number of different values in brackets (NV). The column labeled as "Hybrid" gives the length of individuals (number of genes) with hybrid coding. Finally, the last one shows the length of individuals encoded with natural coding. These lengths were calculated easily from the second and third column. Hybrid

Table 5. Comparing length of Hybrid and Natural Individuals.

Database	NC	ND (NV)	Hybrid	Natural
breast-c	9	-	18	9
bupa	6	-	12	6
cleve	6	7 (19)	31	13
german	7	13 (54)	68	20
glass	9	-	18	9
heart	13	-	26	13
hepatiti	6	13 (26)	38	19
horse-co	7	15 (55)	69	22
Iris	4	-	8	4
Lenses	-	4 (9)	9	4
Mushroom	-	22 (117)	117	22
Pma	8	-	16	8
tic-tac-	9	-	18	9
vehicle	18	-	36	18
vote	-	16 (48)	48	16
wine	13	-	26	13
zoo	-	16 (36)	36	16
Average			34,94	13

coding uses two genes to represent continuous attributes and a number k of genes for discrete ones, k being the number of different values of the attribute. On the other hand, natural coding uses only one gene for each attribute, regardless of its type (continuous or discrete). Thus, the length for hybrid individuals is $2 \times NC + NV$, whereas for natural individuals is $NC + NV$. As we can see, natural coding noticeably decreases the length of individuals. On average, it obtains a reduction greater than 63% regarding the hybrid individuals.

The most important contribution of natural coding is the decrease in size of the search space. This aspect is considered only when the dataset includes continuous attributes. For discrete attributes, natural coding does not reduce the space, since it is based on binary coding. Thus, if Ω_d is the set of values for a discrete attribute, then the size of the search space is $2^{|\Omega_d|}$ for both codings. However, hybrid coding represents continuous attributes with real coding, i.e the genes can take values in an infinite domain. Thus, when there are continuous attributes, the search space with hybrid individuals is also infinite. On the contrary, genes that encode continuous attributes with natural coding can only take values belonging to a finite set of natural numbers. Let N be the number of cutpoints that USD algorithm obtains for an attribute a_i. Let Ω_c be the set of natural numbers that the gene g_i can take according to the cutpoints. Then, the size of the search space for the attribute a_i is the number of elements of Ω_c, that it is given by Equation 3.

$$|\Omega_c| = \binom{N}{2} = \frac{N(N-1)}{2} \tag{3}$$

Thus, the total size of the search space for a dataset with a number M of attributes using hybrid or natural coding is

$$S = \prod_{i=1}^{M} |\Omega_i| \tag{4}$$

where $|\Omega_i|$ is the number of possible values of the attribute a_i. If there are any continuous a_i, such size is infinite for hybrid coding but finite for natural coding.

6 Conclusions

In this paper, natural coding for EA-based decision rules generation is described and tested. This coding transforms the attributes domain (continuous and discrete) in a finite set of natural numbers. Such conversion allows a decrease in size of search space in such a way that the algorithm converges more quickly. Furthermore, natural coding encodes each attribute with only one gene, reducing considerably the length of individuals. The quality of this coding has been tested by applying the same evolutionary learning tool (HIDER) with natural and hybrid coding and by comparing the obtained sets of rules for a set of databases from the UCI Repository. The algorithms with hybrid coding needed 300 generations and a population with 100 individuals in order to obtain a suitable model. Nevertheless, the use of natural coding obtained models with the same accuracy, but a lesser number of rules. Although this is a noteworthy aspect, the most important fact is that such results were obtained with only 70 individuals per population and a number of generations as little as 100. The faster convergence of the algorithm is due to the decreasing in size of search space, since natural coding converts the infinite domain of the continuous attributes to a finite set of natural numbers.

References

1. Jesus S. Aguilar-Ruiz and J.C. Riquelme. Improving the Evolutionary Coding for Machine Learning Tasks. *European Conference on Artificial Intelligence, ECAI'02*, pp. 173–177, IOS Press, Lyon, France, 2002.
2. Jesus S. Aguilar-Ruiz, J.C. Riquelme and M. Toro. Evolutionary Learning of Hierarchical Decision Rules. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, ISSN: 1083-4419, vol. 33, no. 2, pp. 324–331, 2003.
3. J. Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In: *Third International Conference on Genetic Algorithms*, pp. 86–97, 1989.
4. S. Bhattacharyya and G. Koehler. An analysis of non–binary genetic algorithms with cardinality 2^v. *Complex Systems* 8, pp. 227–256, 1994.
5. C. Blake and E. K. Merz. UCI repository of machine learning databases, 1998.
6. P. Clark and R. Boswell. The cn2 induction algorithm. *Machine Learning*, vol. 3, no. 4, pp. 261–283, 1989.

7. P. Clark and R. Boswell. Rule induction with cn2: Some recents improvements. *Machine Learning: Proceedings of EWSL-91*, pp. 51–163, 1991.
8. K. A. DeJong, W. M. Spears, and D. F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 1(13):161–188, 1993.
9. R. Giráldez, Jesus S. Aguilar-Ruiz and J.C. Riquelme. Discretization Oriented to Decision Rule Generation *International Conference on Knowledge-Based Intelligent Information & Engineering Systems*, KES'02 IOS Press, pp. 275–279 September 16–18, Crema, Italy, 2002.
10. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.
11. C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 1(13):169–228, 1993.
12. G. Koehler, S. Bhattacharyya and M. Vose. General cardinality genetic algorithms. *Evolutionary Computation* 5(4), 439–459, 1998.
13. R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The aq15 inductive learning system: An overview and experiments. In *Proceedings of the American Association for Artificial intelligence Conference (AAAI)*, 1986.
14. T. Mitchell. *Machine Learning.* McGraw Hill, 1997.
15. S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 1994.
16. J. R. Quinlan. *C4.5: Programs for Machine Learning.*, Morgan Kaufmann, San Mateo, California, 1993.
17. J. R. Quinlan. See5.0 (http://www.rulequest.com), 1998–2001.
18. N. J. Radcliffe. Genetic Neural Networks on MIMD Computers. Ph. d., University of Edinburgh, 1990.
19. G. Syswerda. Uniform crossover in genetic algorithms. In: *Proceedings of the Third International Conference on Genetic Algorithms.* pp. 2–9, 1989.
20. G. Venturini. SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of European Conference on Machine Learning*, pp. 281–296, 1993.
21. M. Vose and A. Wright. The simple genetic algorithm and the Walsh transform: Part I, theory. *Evolutionary Computation* 6(3), pp. 253–273, 1998.
22. M. Vose and A. Wright. The simple genetic algorithm and the Walsh transform: Part II, the inverse. *Evolutionary Computation* 6(3), pp. 275–289, 1998.
23. D. R. Wilson and T. R. Martinez. Improved Heterogeneous Distance Functions. *Journal of Artificiall Intelligence Research* 6(1), pp. 1–34, 1997.