Natural Encoding for Evolutionary Supervised Learning

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

***Abstract*—Some of the most influential factors in the quality of the solutions found by an evolutionary algorithm (EA) are a correct coding of the search space and an appropriate evaluation function of the potential solutions. EAs are often used to learn decision rules from datasets, which are encoded as individuals in the genetic population. In this paper, the coding of the search space for the obtaining of those decision rules is approached, i.e., the representation of the individuals of the genetic population and also the design of specific genetic operators. Our approach, called "natural coding," uses one gene per feature in the dataset (continuous or discrete). The examples from the datasets are also encoded into the search space, where the genetic population evolves, and therefore the evaluation process is improved substantially. Genetic operators for the natural coding are formally defined as algebraic expressions.**

Experiments with several datasets from the University of California at Irvine (UCI) machine learning repository show that as the genetic operators are better guided through the search space, the number of rules decreases considerably while maintaining the accuracy, similar to that of hybrid coding, which joins the well-known binary and real representations to encode discrete and continuous attributes, respectively. The computational cost associated with the natural coding is also reduced with regard to the hybrid representation.

Our algorithm, HIDER*, has been statistically tested against C4.5 and C4.5 Rules, and performed well. The knowledge models obtained are simpler, with very few decision rules, and therefore easier to understand, which is an advantage in many domains. The experiments with high-dimensional datasets showed the same good behavior, maintaining the quality of the knowledge model with respect to prediction accuracy.

***Index Terms*—Decision rules, evolutionary encoding, supervised learning.**

I. INTRODUCTION

DECISION RULES are especially relevant in problems related to supervised learning. Given a dataset with continuous and discrete features or attributes, and a class label, we try to find a rule set that describes the knowledge within data or classifies new unseen data. When the feature a_i is discrete, the rules take the form of "if $a_i \in \{v_1, \ldots, v_k\}$, then *class*," where the values $\{v_1, \ldots, v_k\}$ are not necessarily all those that the feature can take. When the feature a_j is continuous, typically the rules take the form of "if $a_j \in [l_j, u_j]$ then *class*," where l_j and u_j are two real values belonging to the range of the feature and $l_j < u_j$. For example, let us assume that we have a synthetic dataset that associates the weight (in kilograms) and eye color of a person with whether or not she/he will have a paper accepted in a relevant conference. The dataset is a sequence of tuples such as (60, green, no), (70, black, yes), etc. A rule describing the relationship among attribute values and class might be

$$\text{if weight} \in [56, 70] \text{ and eye color is black then yes.}$$

The search for these rules can be tackled with many different techniques, however, evolutionary algorithms (EAs) present particularly good performance when the search space is complex.

Two critical factors influence the decision rules obtained by an EA: the selection of an internal representation of the search space (*encoding*) and the definition of an external function that assigns a value of goodness to the potential solutions (*evaluation*).

In this work, a particular emphasis is placed on the physical representation of the encoding (the genotype: string of genes possessed by an individual) as compared with the features of the supervised learning problem to be optimized by the EA (the phenotype). The mapping operation between these two representations is also very important to define the search space and to guide the specific genetic operators.

In principle, a single gene can affect several features in the phenotype (mapping one-to-m) or one feature in the phenotype can be controlled by multiple genes (mapping m-to-one). These two situations, named *pleiotropy* and *poligeny* [1], respectively, have a strong presence in naturally evolved systems. However, the one-to-one mapping is the most common in evolutionary systems, which represents a simplification of natural evolution.

Winston [2] suggests that good generators of solutions in a search space should possess the following properties.

- Completeness: they eventually produce all positions in a search space.
- Nonredundancy: they never damage efficiency by proposing the same solution twice.
- Informedness: they use possibility limiting information to restrict the solutions they propose accordingly.

Moreover, a good evolutionary encoding should satisfy the following properties.

- Coherence: it should not be possible to encode an element that has no semantic meaning.
- Uniformity: every element will be represented by the same number of encodings and, if possible, will be unique (*uniqueness*).

This work was supported in part by the Spanish Research Agency CICYT under Grant TIN2004-00159 and in part by Junta de Andalucía (III Research Plan).J. S. Aguilar-Ruiz and R. Giráldez are with the School of Engineering, Pablo de Olavide University, 41013 Seville, Spain (e-mail: jsagurui@upo.es; rgirroj@upo.es).

J. C. Riquelme is with the Department of Computer Science, University of Seville, 41004 Seville, Spain (e-mail: riquelme@lsi.us.es).

- Simplicity: the coding function must have easy application in both directions.
- Locality: small modifications to the values (phenotype) should correspond to small modifications to the hypothetical solutions (genotype).
- Consistency: futile or unproductive codings should not exist .
- Minimality: the length of the coding should be as short as possible.

On the other hand, *internal redundancy* should be avoided. A representation is said to contain *internal redundancy* when not all of the genetic information contained in a chromosome is strictly necessary in order to identify uniquely the solution to which it corresponds. It is also very common to find *degeneracy* in representations. A representation is said to exhibit *degeneracy* when more than one chromosome can represent the same solution. Degeneracy is often detrimental to genetic search [3], because it means that isomorphic forms are allowed. In some problems, the effect of one gene suppresses the action of one or more other genes. This feature of interdependency, called *epistasis*, should also be controlled and minimized.

Therefore, the design of a new representation for EAs is not simple if we wish it to have a good performance. In fact, not only should the encoding preserve most of the properties mentioned above, but new genetic operators should also provide some advantages when the new encodings are used. In this work, we present a new evolutionary encoding to produce decision rules. Our approach, named "natural coding," is a one-to-one mapping, and it only uses natural numbers to encode continuous and discrete features. This coding needs a new definition for the genetic operators in order to avoid the conversion from natural numbers to the values in the original space. These definitions are presented, and as it is shown in this paper, the EA can work directly with the natural coding until the end, when the individuals will be decoded to decision rules.

This paper is organized as follows: in Section II, the work related to binary, integer, and real codings is presented; the motivation of our approach is described in Section III; the *natural coding* is presented in Section IV, together with the genetic operators associated with continuous and discrete attributes; the algorithm is illustrated in Section V, discussing the new evaluation method and the length of individuals; later, we compare our approach to the hybrid coding regarding the length of individuals and to C4.5 and C4.5 Rules to search for statistical differences about the error rate and the number of rules in Section VI; finally, the most interesting conclusions are summarized in Section VII.

II. Related Work

Choosing a good genetic representation is critical for the EA to find a good solution for the problem because the encoding will have much influence on achieving an effective search. Binary, Gray, and floating-point chromosome encodings have been widely used in the literature because they have provided generally satisfactory results. Thus, few alternatives have been analyzed. Nevertheless, there are many ways of associating feature values to genes, which should be discussed in light of finding the best one for specific problems.

Taking the aforementioned properties into account, we are going to analyze the most common representations for EAs, from the perspective of supervised learning. To our knowledge, little theoretical work has been developed in the field of evolutionary encoding. Many authors have approached several codings for specific applications, most of them for optimization or scheduling tasks, clustering, and feature selection. However, in supervised learning, the binary, real and hybrid (binary coding for discrete features and real for continuous) codings have commanded the attention of many researchers.

A number of studies dedicated to EAs, beginning with Holland's [4] made use of binary coding. The simplicity and, above all, the similarity with the concept of Darwinian analogy, have advanced its theoretical use and practical application. Several genetic algorithms-based concept learners apply binary coding to encode features with symbolic domains in order to induce decision rules in either propositional (GABIL [5], GIL [6], COGIN [7]) or first-order form (REGAL [8], [9], and DOGMA [10]). However, binary coding is not the most appropriate for continuous domains.

In general, binary coding has been widely used, for instance, for feature selection [11] and data reduction [12].

In general, assuming that the interaction among attributes is not linear, in principle, the size of the search space is related to the number of genes used. For an individual with L genes, the size of the search space is $\prod_{i=1}^{L} |\Omega_i|$, where Ω_i is the alphabet for the ith gene. Traditionally, the same alphabet has been used for every gene, mostly the binary alphabet, so that $\text{size} = |\Omega|^L$. However, for a continuous feature with range $[l, u]$, if we decide to use length L to encode it, the error in the precision of that attribute would be

$$\text{error} = \frac{u-l}{2^L - 1}. \tag{1}$$

Therefore, if $\delta = \min\{|a_i - a_j| \,|\forall a_i, a_j \in [l, u], i \neq j\}$, the minimum encoding length, to ensure the precision is maintained, would be

$$L = \left\lceil \log_2 \left(1 + \frac{u-l}{\delta} \right) \right\rceil. \tag{2}$$

Taking into account (2), the *quantum* defined by *error* will assure that the length is the least that guarantees a good precision for the mutation operator. However, the *locality* problem is not solved for binary representation. Gray encoding, where two individuals next to each other in the search space differ by only one bit, has been studied in depth [13], [14] and offers some advantages in this realm [15], although it suffers from the same lack of precision as binary coding.

Both binary and Gray representations have some drawbacks when applied to multidimensional, high-precision problems. For example, for ten attributes with domains in the range [0,1], where a precision of three digits after the decimal point is required, the length of an individual is about 100. This, in turn, generates a search space size of about 10^{30}.

For many problems, binary and Gray encoding are not appropriate because a value of 1 bit may suppress the fitness contributions of other bits in the genotype (epistasis) and genetic operators may produce illegal solutions (inconsistency). Some

examples of other specific encodings are: Prüfer numbers to represent spanning trees [16] and integer numbers to obtain hierarchical clusters [17].

In real-valued coding [18], the chromosome is encoded as a vector of floating-point numbers of the same length as the solution vector. Each attribute is forced to be within a desired range and the operators are carefully designed to preserve this requirement. The motivation behind floating-point codings is to move the EA closer to the problem space. This coding is capable of representing quite large domains.

On the contrary, its main theoretical problem is the size of the search space. Since the size of the alphabet is infinite, the number of possible schemes is also infinite and, therefore, many schemes that are syntactically different but semantically similar will coexist (*degeneracy*).

In principle, any value can be encoded with an only (real) gene, but this could be avoided by using a discrete search space, where the mapping is one-to-one. However, instead of mapping real values to real values, we will map natural numbers to real intervals. For continuous domains, the intervals can precisely define a range of values within the domain. With this solution, the size of the search space is finite, and therefore so is the number of schemes as well. However, this search space reduction requires a prior discretization of continuous attributes, so the choice of the discretization method is critical. This idea will be explained in detail in Section IV-B.

The approaches gathered in the bibliography, some of them based on evolutionary strategies, use real coding for machine learning tasks [19] or for multiobjective problems [20]. In SIA [21], real coding is applied to a real-world data analysis task in a complex domain. Finally, a combination of binary and real coding is used in [22] in order to benefit from the advantages of both codings by using the binary approach in discrete domains and real coding for continuous attributes.

Fuzzy coding is another alternative for representing a chromosome. Every attribute consists of a fuzzy set and a set of degrees of membership to each fuzzy set [23]. Some other encoding strategies have been used in EAs. These include trees, matrix encodings, permutation encodings, and structured heterogeneous encodings to name a few.

III. Motivation

The main shortcoming of using real coding to produce decision rules is that any value in the range of the attribute could be used as lower or upper bound of the interval for that attribute condition. For example, the C4.5 tool [25] only takes as possible values for the nodes of the decision trees the midpoints among two consecutive values of an attribute. This idea might be used to encode an individual of the genetic population so that only hypothetically good values will be allowed as conditions over an attribute in the rules. In a similar way, Bonissone *et al.* show in [24] some real-world applications where the knowledge of the problem domain benefits the use of EAs.

To clarify this idea, we will use the dataset shown in Fig. 1, whose features are described in Section I. Observing the attribute weight (the first one), C4.5 would investigate as possible values: 56, 58, 59.5, 61, 63, 66, and 69. In other words, C4.5 is applying a local unsupervised method of discretization [26]

(55, green, yes)
(57, green, no)
(59, black, no)
(60, green no)
(62, black,yes)
(64, blue, no)
(68, black, yes)
(70, black, yes)

Fig. 1. Labeled dataset with one continuous and one discrete attribute.

since the class label is not taken into account in this process. The reduction of the number of values is only determined by the number of equal values for the attribute being considered. This unsupervised discretization is not a good choice to analyze possible limit values (either using entropy or any other criterion) for the intervals [27].

A number of remarkable supervised discretization methods have been included in the bibliography, including Holte's 1R [28] and the method of Fayyad and Irani [29]. In [30], a supervised discretization method, named unparametrized supervised discretization (USD) is presented. This method is very similar to 1R, although it does not need any input parameter. However, as the aim of this method is not to find intervals but cutpoints to be used as limits of further decision rules, we assume that any supervised discretization method would be appropriate for this purpose. As we will see below, if the discretization method produces k cutpoints, then there will be $k * (k-1)/2$ possible intervals for the decision rules.

Our goal consists in observing the class along with the discretization method and decreasing the alphabet size. Following the example in Fig. 1, we can note that it is only necessary to investigate the values 56, 61, 63, and 66, because they are values which produce a change of class. Therefore, this coding allows the use of all the possible intervals defined by every pair of cutpoints obtained by means of discretization, together with the feature range bounds.

In short, if we are able to encode every possible interval and every possible combination of discrete values in such a way that the genetic operators make an efficient search of potential solutions, then the proposed representation will be appropriate for our purpose. Next, we are going to present and discuss this new encoding method, which will disclose interesting properties.

The natural coding leads to a reduction of the search space size, which has a positive influence on the convergence of the EA with respect to the hybrid coding HIerarchical DEcision Rules (HIDER [22]). The prediction accuracy is maintained, while the number of rules is decreased, therefore using less computational resources.

IV. Natural Coding

In this section, we propose a new encoding for EAs in order to find decision rules in the context of supervised learning, together with their genetic operators, which will be presented in two independent subsections: discrete and continuous features, respectively. This coding has been named "natural" because it

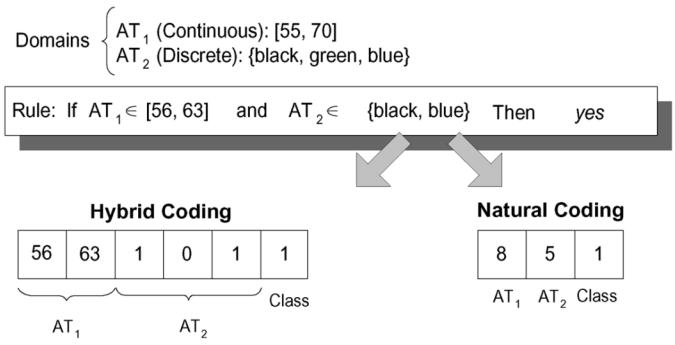

Fig. 2. Hybrid individual versus natural individual.

only uses natural numbers to represent the set of intervals for continuous features and the set of values for discrete ones, which might take part in the decision rules.

Regarding the properties mentioned in Section I, natural coding is complete, nonredundant, and informed. The internal redundancy and degeneracy properties are especially taken into account to perform an effective search for solutions. Moreover, it is coherent, unique, simple, consistent, and minimal. Locality is also satisfied as small variations of the natural numbers correspond to small modifications of the intervals they represent.

Through the text, we will use a very simple dataset in order to explain the application of the genetic operators. This dataset (see Fig. 1) has a continuous attribute (weight in kilograms) with range [55, 70], a discrete attribute (eye color) with values $\{\text{black, green, blue}\}$, and a class (she/he is a candidate to have a paper accepted in a relevant conference) with values $\{\text{yes, no}\}$. Although the semantics of the natural coding have not yet been detailed, Fig. 2 illustrates the length of natural versus hybrid representations. It shows two individuals that represent the same rule. The method to encode these values will be shown later.

Hybrid coding represents a rule comprising the union of two types of genotypes: real coding for the continuous attributes and binary coding for the discrete ones. Each condition associated with a continuous attribute is encoded by two real numbers representing the bounds of the interval for that attribute. Each condition related to a discrete feature is encoded by a vector of binary values. The size of this vector is equal to the number of distinct values of the attribute, so that 1 means that attribute value is present in the condition, and 0, absent. For instance, the rule in Fig. 2 is encoded by using the hybrid coding (left).

The main problem of this encoding is that the search space is very large, since for each continuous feature we have to find two values in the range of the attribute. Another drawback is the length of the individuals, which might be very large when the discrete attributes take many distinct values. Our proposal tries to minimize the search space size and the length of the individuals by assigning only one natural number to each condition regardless of the type of the attributes. Thus, we attempt to limit the search space of valid genotypes and reduce the length of the individuals. When the attributes are discrete, this natural number is the conversion of the aforementioned binary string from the hybrid coding into a decimal number. For continuous features, the natural number is obtained by numbering all the possible intervals the condition can represent in an effective manner. In Fig. 2, a rule is encoded by the natural encoding (right), in contrast with the hybrid encoding (left). In this genotype, the meaning of 8 is the number assigned to the interval [56, 63] in the set of all valid intervals for AT_1, and the number 5 is the integer that represents the binary string 101 for AT_2.

We can note that the natural coding is simpler, since the hybrid coding needs six genes to encode the rule, whereas the natural one encodes it with only three genes. In general, the hybrid one uses two genes for continuous features and $|\Omega_i|$ for discrete ones, where $|\Omega_i|$ is the number of different values that the feature a_i can take. The natural coding gets to minimize the size of individuals, assigning only one gene to each feature.

In general, to choose an encoding and a suitable set of genetic operators is not an easy task. An unacceptable amount of disruption can be introduced to the phenotype as a result of the inappropriate design of genetic operators or the choice of the encoding strategy. Sometimes degeneracy is seen as a beneficial effect to incorporate additional information. However, it is not always a positive phenomenon since the genetic operators might not increase the level of diversity of the next generation. In this case, the implicit parallelism would also be reduced. As a norm, encodings should be designed to naturally suppress redundant encoding forms, and genetic operators should be implemented in such a way that redundant forms do not benefit the operators.

A. Discrete Features

Several systems exist that learn concepts (*concept learners*) and use binary coding for discrete features. When the set of attribute values has many different values, the length of the individual is very high so that a reduction in length might have a positive effect on speeding the algorithm.

In the following discussions, we will assume that a discrete feature is encoded by a single natural number (one gene), and we will analyze how to apply the crossover and mutation operators such that the new values retain the meaning that they would have had with a binary coding. The new value will belong to the interval $[0, 2^{|\Omega|} - 1]$, where $|\Omega|$ is the number of different values of the attribute. With the natural coding for discrete attributes, a reduction of the size of the search space is not obtained by itself, but the individuals are better guided through the search space to look for solutions when the specifically designed "natural genetic operators" are applied.

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 feature must satisfy for an example to be classified. If a value is included in the condition, its corresponding bit is equal to 1, otherwise, it is 0. The natural coding for this gene is the conversion of the binary number into a natural number. Table I shows an example for a discrete feature with three different values: black, green, and blue.

1) Natural Mutation: Following the example in Table I, when a value is selected for mutation, for example, 3 $(011) = \{\text{green, blue}\}$, there are three options: 111, 001, and 010, equivalent to the numbers 7, 1, and 2, respectively. First, we assign the natural number corresponding to each binary number. The mutation of each value would have to be

TABLE I
CODING FOR A DISCRETE ATTRIBUTE

Discrete values			Natural
black	green	blue	Coding
0	0	0	**0**
0	0	1	**1**
0	1	0	**2**
⋮	⋮	⋮	⋮
1	1	0	**6**
1	1	1	**7**

TABLE II
MUTATION VALUES FOR DISCRETE ATTRIBUTES

	0	1	2	3	4	5	6	7
−significant	1	0	3	2	5	4	7	6
	2	3	0	1	6	7	4	5
+significant	4	5	6	7	0	1	2	3

some of the values shown in Table II. For example, possible mutations of 0 are the values 1, 2, and 4.

Definition 1 (Natural Mutation): Let n be the value of a gene of an individual, then the natural mutation of the kth bit of n, denoted by $\mathrm{mut}_k(n)$, is the natural number produced by changing that kth bit

$$\mathrm{mut}_k(n) = (n + 2^{k-1})\%2^k + 2^k \left\lfloor \frac{n}{2^k} \right\rfloor \tag{3}$$

where $k \in \{1, 2, \ldots, |\Omega|\}$; $|\Omega|$ is the number of values of the attribute; $\mathrm{mut}_k(n)$ are the possible mutated values from n; $\%$ is the rest of the integer division; and $\lfloor \cdot \rfloor$ is the integer part.

Example 1: For example, according to Table II, the possible mutation values for $n = 5(101)$ will be 4, 7, and 1

$$\mathrm{mut}_1(5) = (5 + 2^0)\%2^1 + 2^1 \left\lfloor \frac{5}{2^1} \right\rfloor = 4(100)$$
$$\mathrm{mut}_2(5) = (5 + 2^1)\%2^2 + 2^2 \left\lfloor \frac{5}{2^2} \right\rfloor = 7(111)$$
$$\mathrm{mut}_3(5) = (5 + 2^2)\%2^3 + 2^3 \left\lfloor \frac{5}{2^3} \right\rfloor = 1(001).$$

It is worth noting that we do not need to know the binary values, as (3) takes a natural number and provides its natural mutations.

Definition 2 (Mutation Set): Let n be the value of a gene, we define mutation set, $\mathrm{Mut}(n)$, as the set of all valid mutations for a natural value

$$\mathrm{Mut}(n) = \bigcup_{k=1}^{|\Omega|} \mathrm{mut}_k(n) \tag{4}$$

where $\mathrm{mut}_k(n)$ is the natural mutation of the kth binary bit.

Example 2: From Example 1, $\mathrm{Mut}(5) = \{4, 7, 1\}$.

Note that every discrete feature value set represented by a natural number can be mutated by using (4), generating a set of natural numbers which represents new discrete feature value sets. Therefore, binary values do not appear in the population. As shown in Example 2, from the value $5(\{\mathrm{black}, \mathrm{blue}\})$, the possible values are $4(\{\mathrm{black}\}), 7(\{\mathrm{black}, \mathrm{green}, \mathrm{blue}\})$ and $1(\{\mathrm{blue}\})$.

2) Natural Crossover:

Definition 3 (δ-Order Mutation): Let Z be a set of natural numbers, let us define the j-order mutation, and it will be denoted as $[\mathrm{Mut}(Z)]^\delta$, as follows:

$$\begin{aligned}
[\mathrm{Mut}(Z)]^0 &= Z \\
[\mathrm{Mut}(Z)]^1 &= \bigcup_{z \in [\mathrm{Mut}(Z)]^0} (z \cup \mathrm{Mut}(z)) \qquad \vdots \\
[\mathrm{Mut}(Z)]^\delta &= \bigcup_{z \in [\mathrm{Mut}(Z)]^{\delta-1}} (z \cup \mathrm{Mut}(z)).
\end{aligned} \tag{5}$$

Definition 4 (Natural Crossover): Let n_i and n_j be the values of two genes from two individuals for the same feature. The gene of the offspring $\mathrm{Cross}(n_i, n_j)$ will be obtained from the values belonging to the first nonempty intersection between the δ-order mutations. Let $t \geq 0$, and $Q^t = [\mathrm{Mut}(n_i)]^t \cap [\mathrm{Mut}(n_j)]^t$, then

$$\mathrm{Cross}(n_i, n_j) = \{z \in Q^t \,|\, \forall s \geq 0, s < t, Q^t \neq \emptyset, Q^s = \emptyset\}. \tag{6}$$

Example 3: Let us assume that we have the values 6(110) and 3(011). We include the current values into the mutation set since the offspring could be similar to the parents. Thus, 6 mutates to $\{6, 7, 4, 2\}$ and 3 mutates to $\{3, 2, 1, 7\}$. As both genes share 2(010) and 7(111), any of them could be the offspring from 6 and 3. It is possible that the intersection is the parents (for example, for 1 and 3, as the schema is the same for both, 0*1).

Example 4: Lets assume that we have the worst case, the values 1(001) and 6(110), which have no bits in common. The intersection between the first two mutation sets is empty. The process is shown next

$$\begin{aligned}
&[\mathrm{Mut}(\{1\})]^0 = \{1\} \\
&[\mathrm{Mut}(\{6\})]^0 = \{6\} \\
&\mathrm{Then}, Q^0 = \emptyset \\
&[\mathrm{Mut}(\{1\})]^1 = \{0, 1, 3, 5\} \\
&[\mathrm{Mut}(\{6\})]^1 = \{2, 4, 6, 7\} \\
&\mathrm{Then}, Q^1 = \emptyset \\
&[\mathrm{Mut}(\{1\})]^2 = \{0, 1, 2, 3, 4, 5, 7\} \\
&[\mathrm{Mut}(\{6\})]^2 = \{0, 2, 3, 4, 5, 6, 7\} \\
&\mathrm{Therefore}, Q^2 \neq \emptyset \\
&\mathrm{Cross}(1, 6) = \{0, 2, 3, 4, 5, 7\}.
\end{aligned}$$

The intersection will be $\{0, 2, 3, 4, 5, 7\}$, and any of them will be the valid offspring for $\mathrm{Cross}(1, 6)$.

Fig. 3 shows a more explanatory example of the natural genetic operators for the discrete attribute with values $\{\mathrm{white}, \mathrm{red}, \mathrm{green}, \mathrm{blue}, \mathrm{black}\}$. The gene encoded as 11 has the binary code 01011. The block on the right gives the possible mutations, where the changed bit is shown in bold. The gene encoded as 19 follows the same scheme. Therefore, the set of mutations of both 11 and 19 are $\{10, 9, 15, 3, 27\}$ and $\{18, 17, 23, 27, 3\}$, respectively. The crossover between these two genes is taken from the set $\{3, 27\}$.

$$11 = 01011 = \{red, blue, black\} \begin{cases} 01010{=}10{\equiv}\{red, blue\} \\ 01001{=}\ 9 {\equiv}\{red, black\} \\ 01111{=}15{\equiv}\{red, green, blue, black\} \\ 00011{=}\ 3 {\equiv}\{blue, black\} \\ 11011{=}27{\equiv}\{white, red, blue, black\} \end{cases}$$

$$19 = 10011 = \{white, blue, black\} \begin{cases} 10010{=}18{\equiv}\{white, blue\} \\ 10001{=}17{\equiv}\{white, black\} \\ 10111{=}23{\equiv}\{white, green, blue, black\} \\ 11011{=}27{\equiv}\{white, red, blue, black\} \\ 00011{=}\ 3 {\equiv}\{blue, black\} \end{cases}$$

$$Mut(11) = \{10, 9, 15, 3, 27\}$$
$$Mut(19) = \{18, 17, 23, 27, 3\}$$

$$Cross(11, 19) \in \{Mut(11) \bigcup \{11\}\} \bigcap \{Mut(19) \bigcup \{19\}\} = \{10, 9, 15, 3, 27, 11\} \bigcap \{18, 17, 23, 27, 3, 19\} = \{3, 27\}$$

Fig. 3. Example of mutation and crossover operators for a discrete attribute with five values {white, red, green blue, black}.

TABLE III
INTERVALS CALCULATED FOR THE CONTINUOUS ATTRIBUTE WITH RANGE [55,70]. THE BOUNDARY POINTS ARE $\{55, 56, 61, 63, 66, 70\}$. A NATURAL NUMBER IS ASSOCIATED WITH EVERY CORRECT INTERVAL

Cutpoints	56	61	63	66	70
55	$\mathbf{1} \equiv [55, 56]$	$\mathbf{2} \equiv [55, 61]$	$\mathbf{3} \equiv [55, 63]$	$\mathbf{4} \equiv [55, 66]$	$\mathbf{5} \equiv [55, 70]$
56	-	$\mathbf{7} \equiv [56, 61]$	$\mathbf{8} \equiv [56, 63]$	$\mathbf{9} \equiv [56, 66]$	$\mathbf{10} \equiv [56, 70]$
61	-	-	$\mathbf{13} \equiv [61, 63]$	$\mathbf{14} \equiv [61, 66]$	$\mathbf{15} \equiv [61, 70]$
63	-	-	-	$\mathbf{19} \equiv [63, 66]$	$\mathbf{20} \equiv [63, 70]$
66	-	-	-	-	$\mathbf{25} \equiv [66, 70]$

B. Continuous Features

Using binary encoding in continuous domains requires transformations from binary to real for every feature in order to apply the evaluation function. Moreover, when we convert binary into real, the precision might be affected. Ideally, the mutation of the less significant bit of an attribute should include or exclude at least one example from the training set. The real coding seems more appropriate with real domains, simply because it is more natural to the domain. A number of authors have investigated nonbinary EAs theoretically [31]–[35]. In this sense, each gene would be encoded with a float value. Two float values would be needed to express the interval of a continuous feature.

As the range of continuous attributes is infinite, it would be interesting to reduce the search space size, as the computational cost should be lower. This reduction should not have negative influence on the prediction accuracy of the solutions (decision rules) found by the EA. The first step, therefore, consists in diminishing the cardinality of the set of values of the attribute.

1) Reducing the Cardinality: First, we will analyze what intervals inside the range of the attribute tend to appear as intervals for a potential decision rule obtained from the natural coding. As mentioned before, this task could be solved by any supervised discretization algorithm. In [27], it is carried out via an experimental evaluation of various discretization schemes in different evolutionary systems for inductive concept learning, where our tool with natural coding, named HIDER*, was also analyzed. This study showed that HIDER* was robust for any discretization method [27, Table II], although USD [30] turned out to be the most stable discretizer used in that experiment (the last column in [27, Table IV]).

Once the vector indicating the boundaries for the intervals is obtained (*vector of cutpoints*), we assign natural numbers to any possible combination, as shown in Table III.

Example 5: Let us assume from the dataset in Fig. 1 that the output of the discretization algorithm is the vector of cutpoints $\{55, 56, 61, 63, 66, 70\}$ for the attribute weight (note that the upper and lower bounds of the feature range are also included: 55 and 70). We just need to observe the changes of labels when the examples are ordered by the attribute weight (particular situations of this strategy are discussed in [30]). The possible intervals to be generated from those values are shown in Table III. Each interval is identified by a natural number, for example, the interval [61,66] will be referenced by the natural number 14.

Table III shows 6 cutpoints, which can generate 15 possible intervals. The number of intervals defines the size of the alphabet for such attribute and depends on the number of cuts k, exactly $|\Omega| = (\mathsf{k}\,(\mathsf{k} - 1))/(2)$.

In Table III, a natural number (in bold), beginning by 1, from left to right and from top to bottom, is assigned to each interval. These "natural" numbers will help us to identify such intervals later.

2) Transitions: Once the necessary number of cutpoints has been calculated, we know the elements of the new alphabet

Ω. From now, we will analyze the mutation and crossover operators for this encoding. Table III defines the new alphabet $\{1, 2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15, 19, 20, 25\}$.

Definition 5 (Row and Column): Let n be the value of the gene, and let *row* and *col* be the row and the column, respectively, where n is located in Table III. The way in which *row* and *col* are calculated is: (% is the remainder of the integer division)

$$\begin{aligned} \text{row}(n) &= \left\lfloor \frac{n-1}{\mathsf{k}-1} \right\rfloor + 1 \\ \text{col}(n) &= (n-1)\%(\mathsf{k}-1) + 1. \end{aligned} \tag{7}$$

Example 6: Let $n_i = 3$ and $n_j = 20$, and $\mathsf{k} = 6$. Then

$$\begin{aligned} \text{row}(3) = 1 \quad \text{col}(3) = 3 \\ \text{row}(20) = 4 \quad \text{col}(20) = 5. \end{aligned}$$

Therefore, as we can see in Table III, 3 is in row 1 and column 3, and 20 is in row 4 and column 5.

Definition 6 (Boundaries): Let n be the value of the gene, we name boundaries of n to those values from Table III that limits the four possible shifts (one by direction): left, right, up and down, and they will be denoted as *leftb* (left bound), *rightb* (right bound), *upperb* (upper bound), and *lowerb* (lower bound), respectively, and they will be calculated as

$$\begin{aligned} \text{leftb}(n) &= (\mathsf{k}-1)(\text{row}-1) + \text{row} = \mathsf{k} * \text{row} - \mathsf{k} + 1 \\ \text{rightb}(n) &= (\mathsf{k}-1)\text{row} \\ \text{upperb}(n) &= \text{col} \\ \text{lowerb}(n) &= (\mathsf{k}-1)(\text{col}-1) + \text{col} = \mathsf{k} * \text{col} - \mathsf{k} + 1. \end{aligned} \tag{8}$$

Example 7: The number 9 could reach up to 7 to the left, up to 10 to the right, up to 4 to the top and up to 19 to the bottom (see Table III)

$$\begin{aligned} \text{leftb}(9) = 7 \quad \text{rightb}(9) = 10 \\ \text{upperb}(9) = 4 \quad \text{lowerb}(9) = 19. \end{aligned}$$

Definition 7 (Shifts): The left, right, upper, and lower adjacent shifts for a value n will be obtained (if possible) as follows:

$$\begin{aligned} \text{left}(n) &= \max(\text{leftb}(n), n-1) \\ \text{right}(n) &= \min(\text{rightb}(n), n+1) \\ \text{upper}(n) &= \max(\text{upperb}(n), n-\mathsf{k}+1) \\ \text{lower}(n) &= \min(\text{lowerb}(n), n+\mathsf{k}-1). \end{aligned} \tag{9}$$

We define *horizontal* and *vertical* shifts as all the possible shifts for a given row and column, respectively, including n itself

$$\overline{\text{hor}}(n) = \bigcup_{i=1}^{\mathsf{k}-1} \max(\text{leftb}(n), (\mathsf{k}-1)(\text{row}(n)-1) + i) \tag{10}$$

$$\overline{\text{ver}}(n) = \bigcup_{i=1}^{\mathsf{k}-1} \min(\text{lowerb}(n), (\mathsf{k}-1)(i-1) + \text{col}(n)). \tag{11}$$

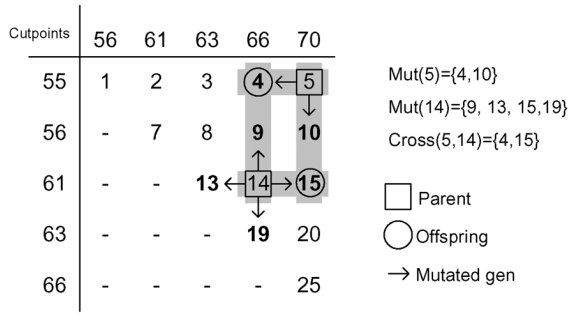

Fig. 4. Mutation and crossover for continuous attributes.

Example 8: From Example 7, the adjacent shifts of 9 will be

$$\begin{aligned} \text{left}(9) = 8 \quad \text{right}(9) = 10 \\ \text{upper}(9) = 4 \quad \text{lower}(9) = 14 \end{aligned}$$

and the horizontal and vertical shifts of 9 will be

$$\overline{\text{hor}}(9) = \{7, 8, 9, 10\} \quad \overline{\text{ver}}(9) = \{4, 9, 14, 19\}.$$

3) Natural Mutation: A mutation consists in selecting a near interval to the one that contains the value n. For example, observing Table III, if the number of cutpoints is $\mathsf{k} = 6$, and $n = 9$, there are four possible mutations $\{4, 8, 10, 14\}$; if $n = 10$, there are three possible mutations $\{5, 9, 15\}$; if $n = 19$, there are two possible mutations $\{14, 20\}$; and finally, if $n = 25$, there is only one possible mutation $\{20\}$.

Definition 8 (Natural Mutation): Let n be the value of the gene, the natural mutation of n, denoted by $\text{Mut}(n)$, is any near value to n by using the shifts and distinct from n

$$\text{Mut}(n) \in \{x \,|\, x \in (\text{mov}(n) - \{n\})\} \tag{12}$$

where $\text{mov}(n) = \text{left}(n) \cup \text{right}(n) \cup \text{upper}(n) \cup \text{lower}(n)$.

Example 9: Thus, $\text{Mut}(2) \in \{\{1, 2, 3, 7\} - \{2\}\}$, i.e., $\text{Mut}(2) \in \{1, 3, 7\}$. Now, one out of the three values could be selected as mutation.

When a gene encodes a continuous feature, the mutation consists in selecting an interval close to it. This is easy to see in Fig. 4, which shows the same example as Table III. For example, gene 14 is the interval [61, 66] and it can be mutated into another interval belonging to the set $\{[56, 66], [61, 63], [61, 70]$ and $[63, 66]\}$ (genes 9, 13, 15, and 19). However, gene 5 can only be transformed into 4 and 10, because there are no genes above or to the right of it.

4) Natural Crossover: Given two parents, the crossover should ideally produce a valid offspring, sharing the genetic material of both parents.

Definition 9 (Natural Crossover): The natural crossover between two values n_i and n_j, denoted by $\text{Cross}(n_i, n_j)$ is obtained as follows:

$$\begin{aligned} \text{Cross}(n_i, n_j) \in ((\overline{\text{hor}}(n_i) \cap \overline{\text{ver}}(n_j)) \\ \cup (\overline{\text{hor}}(n_j) \cap \overline{\text{ver}}(n_i))). \end{aligned} \tag{13}$$

We can observe in Table III that the interesting values are in the intersection between the row and the column where both

values being crossed are placed. Only when the values n_i and n_j are located in the same row or column, the interval will be inside the other.

Example 10: Thus

$$\begin{aligned}\text{Cross}(5,14) \in ((&\{1,2,3,4,5\}\\ &\cap \{4,9,14,19\})\\ &\cup (\{13,14,15\} \cap \{5,10,15,20,25\})),\\ &\text{i.e., Cross}(5,14) \in \{4,15\}.\end{aligned}$$

The general case of crossover between two parents is illustrated in Fig. 4. The possible offspring is formed by those numbers in the intersection between the row and the column of both parents. For example, the crossover between genes 5 and 14 (within squares) generates as offspring genes 4 and 15 (within circles). In Table III, we can see that this offspring makes sense because it uses every boundary from the parents.

V. Algorithm

HIDER is a tool that produces a hierarchical set of rules [22]. When a new example is to be classified, the set of rules is sequentially evaluated according to the hierarchy. 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 it 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 applying the mutation and crossover operators. In each generation, some individuals are selected according to their goodness and they are included in the next population along with their offspring.

The pseudocode of HIDER is shown in Fig. 5. The main algorithm is a typical sequential covering method [36], where the function that produces each rule is an EA. Each call to this function (line 8) generates only one rule that is included in the final set of rules (line 9) and 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 ($\text{epf} = 0$) or a percentage of the training set has been already covered ($\text{epf} \in (0,1]$).

The function *EvoAlg* has a set of examples as its input parameter. It returns a rule that 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 initializing the population, the for-loop repeats the evolutionary process a number of times that 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 generation (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

```
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 × epf
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. 5. Pseudocode of HIDER.

method. Once the loop finishes, the best individual of the last generation is returned.

Equation (14) gives the fitness function ($f(r)$) used by HIDER during the evaluation process. The greater the value, the better the individual is

$$f(r) = N - \text{CE}(r) + G(r) + \text{coverage}(r) \qquad (14)$$

where r is an individual; N is the number of examples being processed; $\text{CE}(r)$ is the class error, i.e., the number of examples belonging to the region defined by the rule r, which do not have the same class; $G(r)$ is the number of examples correctly classified by r; and $\text{coverage}(r)$ gives the size proportion of the search space covered by the rule. A description about the *coverage* factor and its positive influence on the results can be found in [22].

A. Evaluation

A very important aspect of our approach is that the examples of the training file (dataset) are also encoded with natural coding, so that each example of the dataset has the same structure as a rule. If an attribute is continuous, the gene that represents it in an encoded example is the smallest interval that includes the value of the attribute in such example. For example, according to Table III, if the attribute takes the value 64, then the gene is 19, since the smallest interval that includes 64 is [63,66]. However, if the attribute takes the value 65, it will be encoded with the number 19 as well. This means that the original dataset

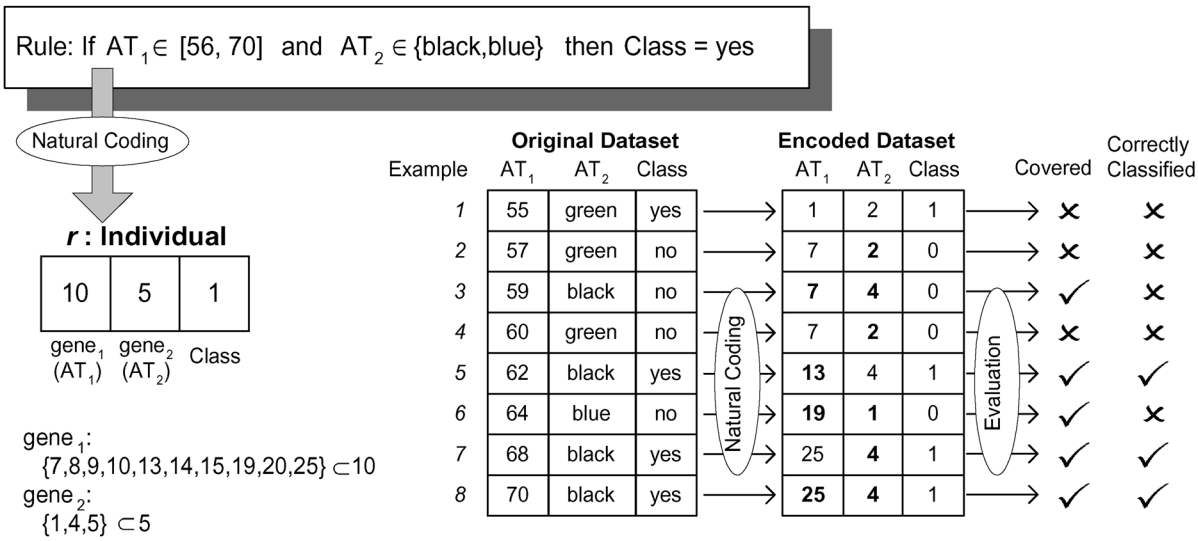

Fig. 6. Example of evaluation.

is reduced before applying the EA, only associating a weight to each example. Thus, a gene that represents a continuous attribute can take only a natural number belonging to the main diagonal of the coding table. For discrete attributes, we will use Table I to encode the values of examples. For example, if the attribute takes the value "*blue*," then the gene is 1. Thus, the example (64, *blue, no*) is encoded as (19,1,0) (see Fig. 6).

The encoding of examples is carried out to speed up the evaluation process, since to decode all of the rules in each evaluation implies higher computational cost. For instance, let r_k be the condition (encoded interval) that a rule establishes for a continuous attribute k, and let e_k be the encoded value that such attribute takes in an example. The row and column of r_k and e_k are $\mathrm{row}(r_k), \mathrm{col}(r_k), \mathrm{row}(e_k)$ and $\mathrm{col}(e_k)$, respectively, in the coding table, and the value e_k fulfils the condition r_k if "$\mathrm{row}(r_k) \leq \mathrm{row}(e_k)$" and "$\mathrm{col}(r_k) \geq \mathrm{col}(e_k)$." Therefore, to know whether or not an example is covered by a rule only consists in comparing four numbers. For discrete attributes, the evaluation is even easier. A condition r_k covers a discrete value e_k (both encoded) if "$r_k \& e_k \neq 0$," "&" being the binary conjunction operator.

In short, an example $e = (e_1, \ldots, e_m, C_e)$ is covered by a rule $r = (r_1, \ldots, r_m, C_r)$ if for each feature $k \in \{1, \ldots, m\}$, the example value is covered by the rule value, $\mathrm{cov}(e_k, r_k)$, as shown in the equation at the bottom of this page.

For example, in Fig. 6, we can see that gene_1 has the value 10, so observing Table III, the values 7, 8, 9, 10, 13, 14, 15, 19, 20, and 25 would correspond to intervals that are inside the interval whose associated value is 10. For the gene_2, with value 5, the values 1, 4, and 5 would satisfy the binary conjunction operation with the value 5.

Regarding the encoded examples, Fig. 6 shows that different original examples are encoded to the same encoded example, for instance, (7,2,0) and (25,4,1). This makes faster the evaluation as the examples (7,2,0) and (25,4,1) would have weight equal to 2.

B. Length of Individuals

A set of datasets from the UCI Repository [37] has been chosen to run the experiments. In Table IV, the length of individuals of the genetic population for the hybrid and natural coding are shown. The first and second columns are the dataset's identifiers and names, respectively. The third shows the number of continuous attributes (NC), if any, for each dataset. Likewise, the next column gives the number of discrete features (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 third and fourth column. The hybrid coding uses two genes to represent continuous attributes and a number k of genes for discrete ones, being k the number of different values of the attribute. On the other hand, the natural coding uses only one gene for each attribute, regardless of its type (continuous or discrete). Thus, the length for hybrid individuals is $2 \times \mathrm{NC} + \mathrm{NV}$, whereas for natural individuals is $\mathrm{NC} + \mathrm{NV}$.

As we can see, the natural coding noticeably decreases the length of individuals. On average, it obtains a reduction greater than 63% with respect to hybrid individuals, which also leads to a search space size reduction. However, the space size reduction is not due to the length, but to the discretization of continuous

$$\mathrm{cov}(e_k, r_k) = \begin{cases} \mathrm{row}(e_k) \leq \mathrm{row}(e_k) \wedge \mathrm{col}(r_k) \geq \mathrm{col}(e_k), & \text{if } a_k \text{ continuous} \\ r_k \& e_k \neq 0, & \text{if } a_k \text{ discrete} \end{cases}$$

TABLE IV
COMPARING LENGTH OF HYBRID AND NATURAL INDIVIDUALS. NC AND ND ARE THE NUMBER OF CONTINUOUS AND DISCRETE FEATURES, RESPECTIVELY. IN BRACKETS, NV IS THE NUMBER OF DISCRETE VALUES

ID	Dataset	NC	ND (NV)	Hybrid	Natural
BC	BREAST CANCER	9	-	18	9
BU	BUPA LIVER DIS.	6	-	12	6
CL	CLEVELAND	6	7 (19)	31	13
GC	GERMAN CREDIT	7	13 (54)	68	20
GL	GLASS	9	-	18	9
HD	HEART DISEASE	13	-	26	13
HE	HEPATITIS	6	13 (26)	38	19
HC	HORSE COLIC	7	15 (55)	69	22
IR	IRIS	4	-	8	4
LE	LENSES	-	4 (9)	9	4
MU	MUSHROOMS	-	22 (117)	117	22
PI	PIMA INDIAN	8	-	16	8
VE	VEHICLE	18	-	36	18
VO	VOTE	-	16 (48)	48	16
WI	WINE	13	-	26	13
ZO	ZOO	-	16 (36)	36	16
	AVERAGE	6.6	6.6 (22.7)	36.0	13.2

attributes. Likewise, it is more natural (and as it will be shown later, more efficient as well) to associate one gene with one attribute, independently of its type.

The binary coding is not appropriate to address this task because the length of individuals is ridiculously large. For instance, by using (2), the individual length for Iris and Cleveland datasets are 22 and 202, respectively. Taking into account that the number of potential solutions would be 2^{22} or 2^{202}, clearly the pure binary coding should be ruled out.

VI. EXPERIMENTS

In order to show the quality of the natural coding, we have designed two kinds of experiments. First, the hybrid coding is compared with the natural coding, to guarantee that the same prediction accuracy can be achieved with less computational cost. Second, the results of the natural coding, C4.5 and C4.5 Rules are statistically analyzed, regarding the error rate and the number of rules, by using two sets of datasets. The first set has 16 datasets with standard size and the second set has 5 large datasets, with several thousands of examples and the number of attributes ranging from 27 to 1558.

A. Natural Coding Versus Hybrid Coding

HIDER and HIDER* have been run with a set of datasets from the UCI Repository. Both tools use the same EA, including the fitness function, although HIDER uses the hybrid coding, in the opposite of HIDER* that uses natural coding. Both were run with the same crossover and mutation parameters, but with a different number of individuals and generations. Table V shows the parameters used in each case. These parameters have great influence on the computational cost, particularly, the number of generations and the population size.

TABLE V
PARAMETERS OF HIDER (HYBRID CODING) AND HIDER* (NATURAL CODING)

Parameter	HIDER	HIDER*
Population Size	100	70
Number of Generations	300	100
Replication	20%	20%
Recombination	80%	80%
Individual Mutation Probability	0.5	0.5
Gene Mutation Probability	$\frac{1}{\|attributes\|}$	$\frac{1}{\|attributes\|}$

Sixteen datasets are used in the experiments, some of which only contain continuous features, others only contain discrete features and the remainder include both types of features. Thus, we can compare the behavior of natural and hybrid coding with both types of features. To measure the performance of each method, a tenfold cross-validation was achieved with each dataset.

The values that represent the performance are the error rate (ER) and the number of rules (NR) obtained. ER is the average number of misclassified examples expressed as a percentage and NR is the average number of rules from the cross-validation. The algorithms were run on the same training sets and the knowledge models tested using the same test sets, so the results were comparable.

In Table VI, the first column shows the datasets 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 an NR for HIDER* with natural coding. The last two columns show a measure of improvement for the error rate (ϵ_{er}) and the number of rules (ϵ_{nr}). The ϵ_{er} coefficient has been calculated by dividing the error rate of HIDER by the corresponding error rate of HIDER*. The same operation has been carried out to obtain ϵ_{nr}, but using 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 to reduce the ER for 7 out of 16 datasets. Nevertheless, on average, it improves HIDER, although such improvement is very small (2%). As regards the number of rules, the results are more significant, since HIDER* obtains 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 smaller number of generations and individuals of genetic population. In particular, HIDER* needed 1/3 of the generations and less than 3/4 of the population size invested by HIDER. This means that HIDER evaluated 30. 000 individuals, in contrast with HIDER*, which only evaluated 7.000, which represents 23% of the initial exploration.

Obviously, this reduction has an important influence on the computational cost. HIDER* needed 4.431 s (1 hour and 14 min,

TABLE VI
COMPARING HIDER AND HIDER*. ER AND NR ARE THE ERROR RATE AND THE NUMBER OF RULES, RESPECTIVELY. ϵ_{er} AND ϵ_{nr} ARE THE IMPROVEMENT OF HIDER* OVER HIDER FOR THE ERROR RATE AND THE NUMBER OF RULES, RESPECTIVELY

	HIDER (hybrid)		HIDER* (natural)		Improvement	
Dataset	ER	NR	ER	NR	ϵ_{er}	ϵ_{nr}
BC	4.3	2.6	4.1	2.0	1.05	1.30
BU	35.7	11.3	37.3	4.2	0.96	2.69
CL	20.5	7.9	25.3	5.9	0.81	1.34
GC	29.1	13.3	27.4	8.0	1.06	1.66
GL	29.4	19.0	35.2	11.7	0.84	1.62
HD	22.3	9.2	21.9	4.3	1.02	2.14
HE	19.4	4.5	16.7	3.7	1.16	1.22
HC	17.6	6.0	20.0	11.1	0.88	0.54
IR	3.3	4.8	3.3	3.2	1.00	1.50
LE	25.0	6.5	25.0	4.5	1.00	1.44
MU	0.8	3.1	1.2	3.5	0.67	0.89
PI	25.9	16.6	25.7	5.1	1.01	3.25
VE	30.6	36.2	33.8	19.7	0.91	1.84
VO	6.4	4.0	4.4	2.2	1.45	1.82
WI	3.9	3.3	8.8	5.6	0.44	0.59
ZO	8.0	7.2	4.0	7.9	2.00	0.91
AVG.	17.6	9.7	18.4	6.4	1.02	1.55

approximately) to complete the tenfold cross-validation for the 16 datasets. Nevertheless, the version with hybrid coding took 12.004 s (about 3 hours and 20 min) in the same standard machine (1.6 GHz CPU, 1 Gb RAM) and for the same tests. Table VII gives detailed time consumed on each dataset. Next to the dataset's identifiers (first column), the second and third columns show the time in seconds taken by HIDER and HIDER*, respectively, to complete the ten executions (training and test phases of each fold of the cross-validation). The last column gives the percentage of time that HIDER* consumed with respect to HIDER, i.e., the third column divided by the second one and multiplied by 100. Note that all these values are smaller than 100%. This fact indicates that HIDER* was faster than HIDER in all of experiments. On average, HIDER*'s runtime was only 35% of the time taken by HIDER, about three times faster.

B. HIDER* *Versus C4.5 and C4.5 Rules*

To show the quality of our approach, the results obtained by HIDER* (natural coding) have been compared with that of C4.5 and C4.5 Rules. The same heterogenous subset of datasets from the UCI Repository, presented in Table IV, was used in the experiments. The cross-validated performance of each method was measured (tenfold cross-validation with each dataset). However, the fact that the results are comparable does not make them significant. Thus, a statistical analysis was carried out, specifically, the Student's t-Test of difference of means with a critical value $\alpha < 0.05$.

TABLE VII
COMPARING THE RUNTIME (IN SECONDS) FOR HIDER AND HIDER*

Dataset	HIDER (hybrid)	HIDER* (natural)	Percent
BC	133	39	29%
BU	160	26	16%
CL	222	73	33%
GE	1832	495	27%
GL	258	71	28%
HD	223	43	19%
HE	104	36	35%
HC	342	275	80%
IR	21	6	29%
LE	5	1	20%
MU	3778	2203	58%
PI	736	95	13%
VE	3862	942	24%
VO	196	43	22%
WI	52	43	83%
ZO	80	40	50%
Average			35.3%

The parameters of EA were initially set (see Table V) and remained constant for all the experiments. This is a very important detail since such parameters could have been specifically set for each dataset and the results would have been substantially better.

The performance of each tool has been measured by means of the error rate (ER) and the number of rules (NR) produced by each method for the 16 datasets, in addition to the standard deviation (σ) of each measurement. The ER is the average number of misclassified examples expressed as a percentage for the tenfold cross-validation. Table VIII gives the results obtained by HIDER*, C4.5, and C4.5 Rules. The first column shows the databases used in the experiments. The next block of two columns gives the error rate $(\mathrm{ER} \pm \sigma)$ and the number of rules $(\mathrm{NR} \pm \sigma)$ obtained by HIDER*. The fourth and fifth columns have the same meaning, in this case for C4.5. The next two columns (HIDER* versus C4.5) show the results of the statistical test with respect to ER and NR. The meaning of the symbols is as follows: "–" denotes that HIDER* obtains a result worse than C4.5; "+" denotes that HIDER* obtains a result better than C4.5; and "•" means that the result is statistically significant (positive or negative). The eighth and ninth columns show the ER and NR for C4.5 Rules and the next two columns the statistical significance of the comparison between HIDER* and C4.5 Rules.

As Table VIII shows, both algorithms tie regarding the error rate, although C4.5 is significantly better in 2 out of 7 datasets. On the contrary, HIDER* obtains better error rate with statistical significance in one dataset. However, as regards the number of rules, HIDER* outperforms C4.5 for all the datasets, where the improvement is significant in 15 out of 16 datasets. The comparison between C4.5 Rules and HIDER* is given next. We can observe that there is also a tie for the error rates, although in two cases HIDER* outperforms C4.5 Rules significantly. For the

TABLE VIII
STATISTICAL ANALYSIS OF RESULTS (STUDENT'S T-TEST OF DIFFERENCE OF MEANS WITH A CRITICAL VALUE $\alpha < 0.05$). SYMBOLS + AND − MEAN THAT HIDER* IS BETTER OR WORSE, RESPECTIVELY. IF THE SYMBOL • APPEARS NEXT, THEN + OR − ARE STATISTICALLY SIGNIFICANT

	HIDER*		C4.5		HIDER* vs. C4.5		C4.5Rules		HIDER* vs. C4.5Rules	
DS	ER± σ	NR± σ	ER± σ	NR± σ	ER	NR	ER± σ	NR± σ	ER	NR
BC	4.1± 2.0	2.0± 0.0	6.3± 3.0	22.9± 3.0	+	+•	4.9± 2.4	9.6± 1.1	+	+•
BU	37.3±11.4	4.2± 0.8	33.7± 9.3	29.7± 5.1	−	+•	32.0± 6.2	11.9± 2.2	−	+•
CL	25.3±10.5	5.9± 0.9	23.5± 7.0	38.3± 5.8	−	+•	25.9±14.7	12.2± 4.6	+	+•
GE	27.4± 3.9	8.0± 1.4	32.9± 4.3	204.2± 8.5	+•	+•	28.8± 3.1	6.2± 2.2	+	−
GL	35.2± 7.8	11.7± 1.6	30.8±11.4	25.8± 2.0	−	+•	18.5± 5.9	15.0± 2.8	−•	+•
HD	21.9± 8.8	4.3± 0.5	25.5± 5.1	33.5± 4.5	+	+•	20.7± 7.0	11.5± 2.0	−	+•
HE	16.7±11.0	3.7± 0.7	19.4± 7.0	15.5± 1.7	+	+•	16.9± 6.1	6.3± 3.6	+	+•
HC	20.0± 7.7	11.1± 1.9	19.0± 7.7	44.4± 3.8	−	+•	17.5± 8.2	5.0± 1.9	−	−•
IR	3.3± 4.7	3.2± 0.4	5.3± 6.9	5.7± 0.5	+	+•	4.7± 7.1	5.0± 0.0	+	+•
LE	25.0±26.4	4.5± 0.9	30.0±21.9	5.2± 0.9	+	+	16.7±22.2	4.1± 0.3	−	−
MU	1.2± 0.6	3.5± 0.5	0.0± 0.0	17.6± 1.0	−•	+•	0.7± 2.3	17.9± 1.9	−	+•
PI	25.7± 3.4	5.1± 0.7	26.1± 5.4	24.4± 8.1	+	+•	29.7± 3.8	8.3± 3.1	+	+•
VE	33.8± 7.4	19.7± 3.3	27.5± 3.6	74.8±10.0	−•	+•	57.6±14.7	4.1± 4.1	+•	−•
VO	4.4± 2.9	2.2± 0.4	6.2± 3.1	15.9± 3.0	+	+•	5.3± 3.7	7.5± 0.7	+	+•
WI	8.8± 4.2	5.6± 0.8	6.7± 7.8	6.5± 0.9	−	+•	6.7± 7.8	5.6± 0.7	−	−
ZO	4.0± 5.2	7.9± 0.9	7.0±10.6	10.9± 1.8	+	+•	29.8±20.7	6.3± 2.0	+•	−•
AVG.	18.4± 7.4	6.4± 1.0	18.7± 7.1	36.9± 3.8			19.7± 8.5	8.5± 2.1		

number of rules, HIDER* improves C4.5 Rules in 11 datasets, being significant in 10 cases. C4.5 Rules provides smaller number of rules in 6 cases, of which only 3 are significant.

On average (last row), HIDER* obtains 18.4% as error rate and 6.4 rules. This is a considerable improvement against C4.5, since its averaged error rate is very similar (18.7%), but it generates a number of rules (36.9) six times greater than HIDER*. C4.5 Rules obtains an error rate of 19.7% and 8.5 rules on average. In summary, HIDER* significantly reduces the number of rules for most of the datasets used in the experiments without damaging the classification accuracy noticeably. Furthermore, C4.5 and C4.5 Rules carry out a posterior pruning of the nodes and rules, i.e., they eliminate those rules that do not provide any benefit to the model. If HIDER* carries out such pruning, the number of rules would be even smaller without significant growth of the error rate, since the deleted rules are those that cover very few examples with respect to the dataset size.

As previously mentioned, HIDER* needs 1 hour and 14 min to complete all the experiments. On the contrary, C4.5 took about 4 min only in the same machine. However, these results are not comparable, since C4.5 is a deterministic algorithm and HIDER* is an EA that applies a stochastic search to find the solutions. In spite of the computational cost, the good outcomes obtained by our proposal indicate that HIDER* can be very useful in non-real-time applications and in realms where the simplicity of the model is a target.

C. Experiments With High-Dimensional Datasets

It has been shown in previous sections that HIDER* provides a good performance in comparison to HIDER, C4.5, and C4.5 Rules. However, although the datasets used in such experiments are common in this type of comparative analysis, they do not seem large enough for a thorough study. In this section, we present a set of experiments in order to show the performance of our approach with five high-dimensional datasets from the UCI Repository [37]. Specifically, the datasets chosen to run the experiments are illustrated in Table IX, where their characteristics are summarized. The first and second columns are the dataset's identifiers and names, respectively. The third column gives the size of the dataset as the number of examples (NE), whereas the fourth is the number of attributes (NA) for each dataset.

TABLE IX
DATASETS' FEATURES. NE AND NA ARE THE SIZE (NUMBER OF EXAMPLES), AND THE NUMBER OF FEATURES, RESPECTIVELY. THE LAST COLUMN GIVES THE NUMBER OF CLASS LABELS

ID	Database	NE	NA	Classes
AD	ADS	3279	1558	2
KR	KR-VS-KP	3196	36	2
MK	MUSK	6598	166	2
SI	SICK	3772	27	2
SP	SPLICE	3190	60	3

HIDER*, C4.5, and C4.5 Rules were run with these datasets by applying a tenfold cross-validation. Nevertheless, the results obtained by C4.5 are not comparable with those produced by the other classifiers. C4.5 produces models with a very low error rate, although with a complexity extremely greater than HIDER* or C4.5 Rules. For instance, for the Musk dataset, C4.5 obtains an average of 118 rules for a error about 4%, whereas HIDER*

TABLE X
RESULTS WITH HIGH-DIMENSIONAL DATASETS. STATISTICAL ANALYSIS OF RESULTS (STUDENT'S T-TEST OF DIFFERENCE OF MEANS WITH A CRITICAL VALUE $\alpha < 0.05$). SYMBOLS + AND − MEAN THAT HIDER* IS BETTER OR WORSE, RESPECTIVELY. IF THE SYMBOL • APPEARS NEXT, THEN + OR − ARE STATISTICALLY SIGNIFICANT

	HIDER*		C4.5Rules		HIDER* vs. C4.5Rules	
DS	ER± σ	NR± σ	ER± σ	NR± σ	ER	NR
AD	5.2± 1.7	7.6± 4.2	3.1± 1.3	12.0± 3.5	−•	+•
KR	5.6± 1.4	3.3± 0.4	9.4± 1.8	8.4± 0.9	+•	+•
MK	11.4± 2.2	10.7± 1.4	11.7± 2.7	10.2± 9.7	+	−
SI	1.8± 0.6	3.0± 0.0	4.4± 1.7	3.7± 0.7	+•	+•
SP	17.9± 2.7	7.2± 1.1	18.8± 13.3	23.2± 23.5	+	+•
Average	8.4± 1.7	6.4± 1.4	9.5± 4.2	11.5± 7.7		

and C4.5 Rules obtain around 11% as error rate but only 11 rules approximately. The case of the Splice dataset is even more meaningful, where C4.5 reduces the error rate half with respect to HIDER*, but with 253 rules on average, i.e., 37 times greater than the 7 rules produced by HIDER*. Although C4.5 reduces the error rate significantly, the size of the decision trees does make them unintelligible.

Table X gives the results obtained by HIDER* and C4.5 Rules. The performance of each tool has been measured the same way as in previous experiments, i.e., by means of the error rate (ER) and the number of rules (NR). The statistical analysis was also carried out and the results are presented with the same symbols, as in Table VIII. Since these datasets have a great number of attributes and the size is also large, some parameters of EA were changed with respect to those shown in Table V. In particular, the population size was set to 100 individuals and the number of generations to 300, as previously used HIDER.

In general, HIDER* has a good performance. Regarding the error rate, HIDER* is better in four datasets, being significant in two cases. C4.5 Rules obtains a relevant reduction of the error in one dataset (AD), although increasing slightly the number of rules. As regards the number of rules, HIDER* outperforms C4.5 Rules in four datasets, all of them with statistical significance. On average, HIDER* reduces both measurements, the error rate (8.4% against 9.5%) and the number of rules (6.4 against 11.5).

In some high-dimensional domains, it is extremely important to produce understandable models, with very few rules. HIDER* shows an excellent performance when the size and dimensionality of the dataset are high, mainly with respect to the complexity of the model.

VII. CONCLUSION

In this paper, a new encoding method for EAs in supervised learning is presented. This method transforms every attribute domain into a natural number domain, for continuous and discrete attributes. The population will therefore have only natural numbers. The genetic operators (mutation and crossover) are defined as algebraic expressions in order to work efficiently with this new search space, and they use no conversions from the original feature domains to the natural number domains, but the EA works from the beginning to the end with natural numbers.

Another important advantage of HIDER* is that all the examples from the database are encoded into the search space, making the evaluation process very fast.

The natural coding for EA-based decision rules generation is described and tested by using tenfold cross-validation with 21 datasets from the UCI Repository (five of them very large). The quality of this coding has been tested by applying the same evolutionary learning tool with natural (HIDER*) and hybrid (HIDER) coding, also improving the computational cost. HIDER* has been compared with C4.5 and C4.5 Rules in order to find statistical differences and the experimental results show an excellent performance, mainly with respect to the number of rules, maintaining the quality of the acquired knowledge model.

ACKNOWLEDGMENT

The authors are very grateful to the reviewers for helpful comments and constructive criticisms.

REFERENCES

[1] D. B. Fogel, "Phenotypes, genotypes and operators in evolutionary computation," in *Proc. 2nd IEEE Int. Conf. Evol. Comput.*, Perth, Australia, 1995, pp. 193–198.

[2] Winston, *Artificial Intelligence*, 3rd ed. Reading, MA: Addisson-Wesley, 1992.

[3] N. J. Radcliffe, "Genetic set recombination and its application to neural network topology optimization," *Neural Comput. Appl.*, vol. 1, no. 1, pp. 67–90, 1993.

[4] J. H. Holland, "Adaptation in natural and artificial systems," Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, 1975.

[5] K. A. DeJong, W. M. Spears, and D. F. Gordon, "Using genetic algorithms for concept learning," *Mach. Learn.*, vol. 1, no. 13, pp. 161–188, 1993.

[6] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Mach. Learn.*, vol. 1, no. 13, pp. 169–228, 1993.

[7] D. P. Greene and S. F. Smith, "Using coverage as a model building constraint in learning classifier systems," *Evol. Comput.*, vol. 2, no. 1, pp. 67–91, 1994.

[8] A. Giordana and F. Neri, "Search-intensive concept induction," *Evol. Comput. J.*, vol. 3, no. 4, pp. 375–416, 1996.

[9] F. Neri and L. Saitta, "An analysis of the universal suffrage selection operator," *Evol. Comput. J.*, vol. 4, no. 1, pp. 89–109, 1996.

[10] J. Hekanaho, "GA-based rule enhancement concept learning," in *Proc. 3rd Int. Conf. Knowl. Discovery Data Mining*, Newport Beach, CA, 1997, pp. 183–186.

[11] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn, and A. K. Jain, "Dimensionality reduction using genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 4, no. 2, pp. 164–171, Apr. 2000.

[12] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study," *IEEE Trans. Evol. Comput.*, vol. 7, no. 6, pp. 561–575, Dec. 2003.
[13] R. Caruana and J. D. Schaffer, "Representation and hidden bias: Gray versus binary codign for genetic algorithms," in *Proc. Int. Conf. Mach. Learn.*, 1988, pp. 153–161.
[14] R. Caruana, J. D. Schaffer, and L. J. Eshelman, "Using multiple representations to improve inductive bias: Gray and binary coding for genetic algorithms," in *Proc. Int. Conf. Mach. Learn.*, 1989, pp. 375–378.
[15] U. K. Chakraborty and C. Z. Janikow, "An analysis of gray versus binary encoding in genetic search," *Inf. Sci.*, no. 156, pp. 253–269, 2003.
[16] F. Rothlauf and D. Goldberg, "Prufer numbers and genetic algorithms," *PPSN*, pp. 395–404, 2000.
[17] J. A. Lozano and P. Larrañaga, "Applying genetic algorithms to search for the best hierarchical clustering of a dataset," *Pattern Recogn. Lett.*, vol. 20, no. 9, pp. 911–918, 1999.
[18] F. Herrera, M. Lozano, and J. L. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for the behavior analysis," *Arti. Intell. Rev.*, vol. 12, pp. 265–319, 1998.
[19] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," *Foundations of Genetic Algorithms-2*, pp. 187–202, 1993.
[20] K. Deb and A. Kumar, "Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems," *Complex Syst.*, vol. 9, pp. 431–454, 1995.
[21] G. Venturini, "SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts," in *Proc. Eur. Conf. Mach. Learn.*, 1993, pp. 281–296.
[22] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "Evolutionary learning of hierarchical decision rules," *IEEE Trans. Syst., Man, Cybern., Part B*, vol. 33, no. 2, pp. 324–331, 2003.
[23] S. K. Sharma and G. W. Irwing, "Fuzzy coding of genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 4, pp. 344–355, Aug. 2003.
[24] P. P. Bonissone, R. Subbu, N. Eklund, and T. R. Kiehl, "Evolutionary algorithms + domain knowledge = real-world evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 256–280, Jun. 2006.
[25] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
[26] D. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Proc. 12th Int. Conf. Mach. Learn.*, 1995, pp. 194–202.
[27] J. S. Aguilar-Ruiz, J. Bacardit, and F. Divina, "Experimental evaluation of discretization schemes for rule induction," in *Genetic and Evolutionary Computation*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, Jun. 2004, pp. 828–839.
[28] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Mach. Learn.*, vol. 11, pp. 63–91, 1993.
[29] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous valued attributes for classification learning," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, 1993, pp. 1022–1027.
[30] R. Giráldez, J. S. Aguilar-Ruiz, and J. C. Riquelme, "Discretization oriented to decision rules generation," in *Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies*. Amsterdam, The Netherlands: IOS-Press, 2002, pp. 275–279.
[31] J. Antonisse, "A new interpretation of schema notation that overturns the binary encoding constraint," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 86–97.
[32] S. Bhattacharyya and G. J. Koehler, "An analysis of non-binary genetic algorithms with cardinality 2^v," *Complex Syst.*, vol. 8, pp. 227–256, 1994.
[33] G. J. Koehler, S. Bhattacharyya, and M. D. Vose, "General cardinality genetic algorithms," *Evol. Comput.*, vol. 5, no. 4, pp. 439–459, 1998.
[34] M. D. Vose and A. H. Wright, "The simple genetic algorithm and the Walsh transform: Part I, Theory," *Evol. Comput.*, vol. 6, no. 3, pp. 253–273, 1998.
[35] M. D. Vose and A. H. Wright, "The simple genetic algorithm and the Walsh transform: Part II, The inverse," *Evol. Comput.*, vol. 6, no. 3, pp. 275–289, 1998.
[36] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
[37] C. Blake and E. K. Merz, "UCI repository of machine learning databases," Univ. California, Irvine, CA, 1998.

Jesús S. Aguilar-Ruiz received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Seville, Seville, Spain, in 1992, 1997, and 2001, respectively.

He is an Associate Professor of Computer Science at Pablo de Olavide University, Seville, Spain. He has been member of the program committee of several international conferences, and reviewer for relevant journals. His areas of research interest include evolutionary computation, data mining and bioinformatics.

Raúl Giráldez received the B.Eng., M.S.Eng., and Ph.D. degrees in computer science from the University of Seville, Seville, Spain, in 1998, 2000, and 2004, respectively.

He is an Assistant Professor at Pablo de Olavide University, Seville, Spain. His areas of research interest include evolutionary algorithms, knowledge discovery, and bioinformatics. He has been member of the program committee of several international conferences related to this areas of research.

Dr. Giráldez received the Doctoral Dissertation Award from the University of Seville.

José C. Riquelme received the M.Sc. degree in mathematics and the Ph.D. degree in computer science from the University of Seville, Seville, Spain, in 1986 and 1996, respectively.

He is an Associate Professor at the University of Seville. He conducts research in genetic programming, feature selection, and data mining. He has served as a member of the program committee of several conferences related to evolutionary computation.