Evolutionary Learning of Hierarchical Decision Rules

Jesús S. Aguilar-Ruiz, José C. Riquelme, and Miguel Toro, *Member, IEEE*

***Abstract*—This paper describes an approach based on evolutionary algorithms, hierarchical decision rules (HIDER), for learning rules in continuous and discrete domains. The algorithm produces a hierarchical set of rules, that is, the rules are sequentially obtained and must be, therefore, tried in order until one is found whose conditions are satisfied. Thus, the number of rules may be reduced because the rules could be inside one another. The evolutionary algorithm uses both real and binary coding for the individuals of the population. We have tested our system on real data from the UCI Repository, and the results of a ten-fold cross-validation are compared to C4.5s, C4.5Rules, See5s, and See5Rules. The experiments show that HIDER works well in practice.**

***Index Terms*—Decision rules, decision trees, evolutionary algorithms (EAs), supervised learning.**

I. Introduction

SUPERVISED learning is used when the user knows the outcomes of the data samples and wants to predict the outcome of a new unseen instance. An algorithm carries out the prediction (classification) and can produce knowledge by using a suitable data structure. Some techniques, like nearest neighbor searching or neural networks, can classify an instance, but cannot obtain the knowledge structure from the information stored in the database. However, other techniques produce sets of rules with a specific structure: decision trees, decision lists, or simply, a set of rules. In general, when a rule-based framework is used to express the acquired knowledge, this is often called decision rules. Such rules can subsequently be used both to infer properties of the corresponding categories and to classify other, previously unseen, examples from the original space.

Decision trees are a particularly useful technique in the context of supervised learning because they perform classification by a sequence of tests whose semantics are intuitively clear and easy to understand. Some tools, like C4.5 [1], construct decision trees selecting the best attribute by using a statistical test to determine how well it alone classifies the training examples. This sort of decision tree may be called axis-parallel, because the tests at each node are equivalent to axis-parallel hyperplanes in such space. On the other hand, other techniques build oblique decision trees, such as OC1 [2], that tests a linear combination of the internal attributes at each node, so that these tests are equivalent to hyperplanes at an oblique orientation to the coordinate axes.

Some algorithms construct a *decision list* [3], a set of rules which is ordered according to some heuristic measure and stored as a list. The future examples are classified by the first rule that matches it in the list. For example, CN2 [4] induces an ordered list of decision rules from examples using entropy as its search heuristic. Some methods, including the "unordered decision list" version of CN2 [5] or the aq-based systems [6] generates a decision rule for each class in turn.

Evolutionary algorithms (EAs) are a family of computational models inspired by the concept of evolution. These algorithms employ a randomized search method to find solutions to a particular problem [7]. This search is quite different from the other learning methods mentioned above. An EA is any population-based model that uses selection and recombination operators to generate new sample examples in a search space [8]. The EA search can move much more abruptly, replacing a parent individual with an offspring less likely to fall into the same kind of local minima which can happen with the other methods. EAs have been used in a wide variety of optimization tasks [9], [10] including numerical optimization and combinatorial optimization problems, although the range of problems to which EAs have been applied is much broader. The main task in applying EAs to any problem consists in selecting an appropriate representation (coding) and an adequate evaluation function (fitness).

In classical EAs the members of the population (typically maintaining a constant-size) are represented as fixed-length strings of binary digits. The length of the strings and the population size are completely dependent on the problem. The population simulates nature's behavior, since the relatively "good" solutions produce offspring which replace ones that are relatively "worse," retaining many of the features of their parents. The estimate of the quality of a solution is based on a fitness function, which determines how good an individual within the population in each generation is. New individuals (offspring) for the next generation are formed by using (normally) two genetic operators: crossover and mutation. Crossover combines the features of two individuals to create several (commonly two) individuals. Mutation operates by randomly changing several components of a selected individual.

Genetic based searching algorithms for supervised learning, as genetic algorithm batch incremental leaner (GABIL) [11], or genetic-based inductive learning (GIL) [12], do not easily handle numeric attributes because the method of encoding all possible values would lead to very long rules in the case or real-valued attributes. Concretely, GABIL and GIL are so-called "concept learners" because they are designed for discrete domains. Other approaches, as supervised inductive algorithm (SIA) [13], have been motivated by a real world

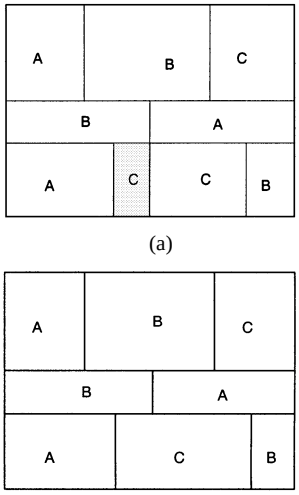

Fig. 1. Comparison between (a) C4.5 and (b) HIDER in the case of nonnested regions. C4.5 needs to split the space more times.

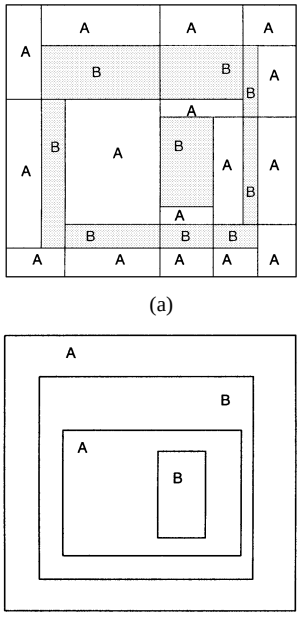

Fig. 2. Comparison between (a) C4.5 and (b) HIDER in the case of nested regions. C4.5 needs much more regions than HIDER.

data analysis task in a complex domain, with continuous and discrete attributes. Recent works use hybrid techniques in which EAs play an important role, such as inducing decision trees [14] or coping with the problem of small disjuncts [15].

The aim of our research was to obtain a set of rules by means of an evolutionary algorithm to classify new examples in the context of supervised learning. With our approach, hierarchical decision rules (HIDER), we try to efficiently handle continuous and discrete attributes. HIDER obtains one rule from each run of the evolutionary algorithm, including this rule in the set of rules. Once the algorithm has finished, these rules are applied to classify new examples in the same order that they were obtained, maintaining its order-dependency. The justification of this method will be discussed in Section II. The characteristics of our approach are presented in Section III, where the coding, the algorithm, the selected fitness function, and a particular aspect named *generalization*, are detailed. Section IV shows the experiments, the results, and their analysis. In Section V, conclusions are summarized, which motivates some of the future works presented in Section VI.

II. Motivation

Two artificial two-dimensional (2-D) datasets will be used to clarify the motivation of our approach. The way in which C4.5 splits the space is depicted in Fig. 1. The figures within the circles describe the level on the tree where the tests (nodes) over these attributes are placed. See the region labeled as B on the bottom-left corner of Fig. 1. C4.5 divides region B into two parts, however, we thought that the complete region should be covered by only one rule. This fact motivates us to design an algorithm, HIDER, which is able to discover such rule.

HIDER is quite different because it does not divide the space by an attribute, but sequentially extracts regions from the space itself. This permits obtaining *pure* regions, i.e., all examples having the same class label. As illustrated in Fig. 1, the region labeled as B on the bottom-left corner is discovered by HIDER.

For another artificial 2-D dataset, Fig. 2 shows the classification that C4.5 gives. Nevertheless, as illustrated in Fig. 2, the quality of the rule set would be improved if the algorithm finds rules within others. The most evident feature, graphically observed in Fig. 2, is the reduction of the number of rules because of the overlapping rules. This characteristic motivates us to use hierarchical decision rules instead of independent (unordered) decision rules.

In short, the obtaining of larger regions without damaging the prediction accuracy, and the discovery of regions within others, are the two main goals which have motivated the development of HIDER. In particular, our algorithm has been successfully applied to a real problem: the generation of decision rules to estimate software development projects [16]. These rules will help the project manager keep the project within cost, quality, and duration goals. It is important to note, that for this real domain, the interest resides in obtaining rules for just one labeled class (good projects from the dataset). This makes our approach more flexible in comparison to other techniques that generate decision rules for all the classes, like C4.5.

III. HIDER

HIDER uses an EA to search for the best solutions and produces a hierarchical set of rules. According to the hierarchy, an example will be classified by the ith rule if it does not match the conditions of the $(i-1)$th preceding rules. The rules are obtained sequentially until the space is totally covered. The behavior is similar to a *decision list* [3]. As mentioned in [5], the meaning of any single rule is dependent on all the other rules which precede it in the rule list, so it might be a problem for the expert to understand if many rules are present. However, in many areas, the rule set is not used to understand the information stored in the database, but used to classify new unseen instances. In this sense, since HIDER generally obtains first rules containing more examples, the number of attributes needed to test the final set of rules decreases. HIDER is designed on the

if conditions **then** class
else if conditions **then** class
else if conditions **then** class
.....
else "unknown class"

Fig. 3. Hierarchical set of rules.

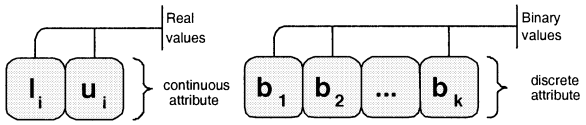

Fig. 4. Continuous (left) and discrete (right) attributes.

basis of accuracy, but the incremental construction favors the understandability of the provided rule set.

We extend the concept of decision lists to continuous domains. When we want to learn rules in the context of continuous attributes, we need to extend the concept of decision lists in two ways. First, to adapt the Boolean functions to interval functions; and secondly, to represent many classes instead of the true and false values (positives and negatives examples). For each continuous (real) attribute a_i, we obtain the boundary values called l_i and u_i (lower and upper bounds, respectively) which define the space R_i (range of the attribute i). These intervals allow us to include continuous attributes in a decision list. A decision list has a last constant function *true*, i.e., examples not covered by any rules will have a defined class label, commonly that of the majority class. We could interpret this last function as an unknown function, that is, we do not know which class the example belongs to. Therefore, it may be advisable to say "unknown class" instead of making an erroneous decision. From the point of view of the experiments, when no induced rules are satisfied, "unknown class" will be considered as an error.

The structure of the set of rules will be as shown in Fig. 3.

As mentioned in [17], one of the primary motivations for using real-coded EAs is the precision to represent attribute values, and the other is the ability to exploit the continuous nature of functions of continuous attributes. We implemented our first versions with binary-coded genetic algorithms, but realized that real-coded EAs are more efficient in time and quality of results [18].

Before an EA can be run, a suitable *coding* for the problem must be devised. We also require a *fitness function*, which assigns a figure of merit to each coded solution. During the run, parents are *selected* for reproduction, and *recombined* to generate *offspring*. These aspects are described below.

A. Coding

In order to apply EAs to a learning problem, we need to select an internal representation of the space to be searched and to define an external function that assigns fitness to candidate solutions. Both components are critical for the successful application of the EAs to the problem of interest.

Information on the environment comes from a data file, where each example has a class and number of attributes. We have to codify that information to define the search space, which normally will be dimensionally greater, since the length of the individual will be greater than the number of attributes. Each attribute will be formed by several components in the search space, depending on the specific representation.

In our first approaches, we studied other EA-based classifiers with binary coding. These are generally used as concept learners, where coding assigns a bit to each value of the attribute, i.e., every attribute is symbolic (GABIL [11] and GIL [12] are two very well-known systems). For example, an attribute with three possible values would be represented by three bits. A value of one in a bit indicates that the value of the attribute is present so that several bits could be active for the same attribute. This coding is appropriate for symbolic domains. However, it is very difficult to use it in continuous domains, because the number of elements in the alphabet is very large, prohibiting a complete search.

Using binary coding in continuous domains requires transformations from binary to real for every attribute in order to apply the evaluation function. Moreover, when we convert binary into real, the precision is lost, so that we have to find the exact number of bits to eliminate the difference between any two values of an attribute. This ensures that a mutation of the less significant bit of an attribute will include, or exclude, at least one example from the training set.

Nevertheless, the real coding is more appropriate with real domains, simply because it is more natural to the domain. A number of authors have investigated nonbinary evolutionary algorithms theoretically [19]–[23]. In this work, real coding is adopted to efficiently handle continuous domains and axis-parallel representations. For example, in [24], other representations are explored, such as, rotated hyperrectangles and hyperellipses.

The representation for continuous and discrete attributes is shown in Fig. 4, where l_i and u_i are values representing an interval for the continuous attribute; b_k are binary values indicating whether the value of the discrete attribute is active or not. A last value (omitted in the figure) is for the class. All the individuals within the population will have this encoding, two genes for continuous attributes and the number of discrete values define the length for discrete attributes.

The number of classes determines the set of values to which it belongs, i.e., if there are five classes, the value will belong to the set $\{0, 1, 2, 3, 4\}$. Each rule will be obtained from this representation, but when $l_i = \min(a_i)$, or $u_i = \max(a_i)$, where a_i is an attribute, the rule will not have that value. For example, in the first case the rule would be $[-, v]$ and in the second case $[v, -]$, v being any value within the range of the attribute. If both values are equal to the boundaries, then the rule $[-, -]$ arises for that attribute, which means that it is not relevant because either of the attribute's values will be covered by the whole range of that attribute ($[-, -]$). Under these assumptions, some attributes might not appear in the set of rules. In the same way, when every discrete value is active, that attribute does not appear in the rule.

B. Algorithm

The algorithm is a typical sequential covering EA [25]. It chooses the best individual of the evolutionary process, trans-

```
Procedure HIDER(E, R)
  R := ∅
  n := |E|
  while |E| > n × epf
    r :=EvoAlg(E)
    R := R ⊕ {r}
    E := E − {e ∈ E | e ⊆ Δ_r}
  end while
end HIDER

Function EvoAlg(E)
  i := 0
  P_0 :=Initialize()
  Evaluation(P_0, i)
  while i < num_generations
    i := i + 1
    for j ∈ {1, ..., |P_{i−1}|}
      x̄ :=Selection(P_{i−1}, i, j)
      P_i := P_i+Recombination(x̄, P_{i−1}, i, j)
    end for
    Evaluation(P_i, i)
  end while
  return best_of(P_i)
end EvoAlg
```

Fig. 5. Pseudocode of HIDER.

forming it into a rule which is used to eliminate data from the training file [13]. In this way, the training file is reduced for the following iteration. HIDER searches for only one rule among the possible solutions, which compared to the algorithms based on the Michigan and Pittsburgh approaches, reduces the search space, even when several searches must be performed if several rules are to be learned.

The pseudocode of HIDER is shown in Fig. 5. The algorithm is divided in two parts: the procedure HIDER, which constructs the hierarchical set of rules and the function EvoAlg, which obtains one rule every time it is run. Initially, the set of rules R is empty, but in each iteration a rule is included (operator $\oplus$) in R; E is the training file and n is the number of remainder examples that have not been covered yet (exactly $|E|$ at the beginning). In each iteration, the training file E is reduced (operator $-$), eliminating those examples that have been covered by the description of the rule r (Δ_r), i.e., the left-hand side of the rule, independently of its class. A parameter epf, called *examples pruning factor*, controls the number of examples that will not be covered during the process (ranging from 1%-5%). This factor ensures that rules covering few examples have not been generated. Some authors have pointed out that these rules are undesirable, especially with noise in the domain [5], [26]. The termination criterion is reached when more examples to cover do not exist, depending on the epf. For the trials, we have set the epf to 0.

The evolutionary algorithm is run each time to discover one rule. The method of generating the initial population (initialize) consists in randomly selecting an example from the training file for each individual of the population. Afterwards, an interval to which the example belongs is obtained. For example, let L_i and U_i be the lower and upper bounds of the attribute i; then, the range of the attribute is $U_i - L_i$; next, we randomly choose an example $(v_{a_1}, \ldots, v_{a_i}, \ldots, v_{a_m}, \text{class})$ from the training file, where m is the number of attributes; at last, a possible individual of the population could thus be $(\ldots, v_{a_i} - ((U_i - L_i)/N)k_1, v_{a_i} + ((U_i - L_i)/N)k_2, \ldots, \text{class})$, where v_{a_i} is a value for the attribute i; k_1 and k_2 are random values belonging to $[0, N/C]$ (N is the size of the training data; C is the number of different classes; and class is the same of that of the example). This initialization assures that at least one example from the dataset is included in the potential rule. The value of the attribute v_{a_i} is used to create a valid interval, although every time with different width, due to the random values k_1 and k_2. For discrete attributes, the individual has as many positions as different values for the attribute, although we assure that at least the same active value of the example will remain active in the individual.

The evolution module includes elitism: the best individual of every generation is replicated to the next one ($j = 1$, see in Fig. 5 the loop controlled by the variable j). A set of children (from $j = 2$ to $j = |P_{i-1}|/2$) is obtained from copies of randomly selected parents, generated by their fitness values and using the roulette wheel selection method. The remainder individuals [from $j = (|P_{i-1}|/2) + 1$ to $j = |P_{i-1}|$] are formed by means of crossovers (recombination). Since half of the new population is created by applying the crossover operator, the probability of crossover is 0.5 and the probability of selecting an individual for crossing depends on its fitness value. These individuals can be mutated (recombination) later and only the individual from the elite will not be mutated. The evaluation function (evaluation) assigns a value of merit to each individual.

1) Crossover: Wright's linear crossover operator [27] creates three offspring: treating two parents as two points, p_1 and p_2; one child is the midpoint of both, and the other two lie, on a line determined by $(3/2)p_1 - (1/2)p_2$ and $-(1/2)p_1 + (3/2)p_2$. Radcliffe's flat crossover [28] chooses values for an offspring by uniformly picking values between the two parents values, inclusively. Eshelman and Schaffer [17] use a crossover operator that is a generalization of Radcliffe's which is called blend crossover (BLX-α). It uniformly picks values that lie between two points that contain the two parents, but may extend equally on either side determined by a user specified EA-parameter α. For example, BLX-0.1 picks values from points that lie on an interval that extends $0.1I$ on either side of the interval I between the parents. Logically, BLX-0.0 is the Radcliffe's flat crossover.

Our crossover operator is an extension of Radcliffes's adapted to individuals coded as intervals. Let $[l_i^j, u_i^j]$ and $[l_i^k, u_i^k]$ be the intervals of two parents, j and k, for the same attribute i. From these parents one child $[l, u]$ is generated by selecting values that satisfy the expression: $l \in [\min(l_i^j, l_i^k),$

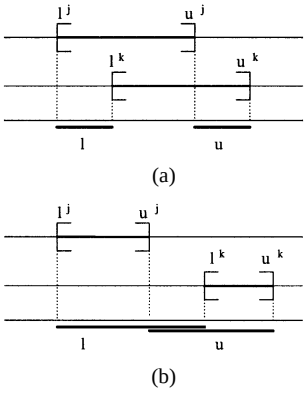

Fig. 6. Crossover situations. (a) valid. (b) invalid.

$\max(l_i^j, l_i^k)]$ and $u \in [\min(u_i^j, u_i^k), \max(u_i^j, u_i^k)]$. This type of crossover could produce two situations which are illustrated in Fig. 6(a) and (b). When the intersection of two intervals is not empty, as shown in Fig. 6(a), the new interval $[l, u]$ is clearly obtained. However, a different situation is produced when the intersection is empty [Fig. 6(b)], because l could be greater than u. In this case, the offspring is rejected (experiments showed better results in this way). When the attribute is discrete, the crossover operator is like a uniform crossover [29].

2) Mutation: Mutation is applied to continuous attributes as follows: if the randomly selected location (gen) is l_i or u_i, then a small value is subtracted or added, depending on whether it is the lower or the upper boundary, respectively. The small value in this work is the smallest heterogeneous overlap-Euclidean metric [30] among any two examples belonging to the dataset.

In the case of discrete attributes, mutation changes the value from 0 to 1, or vice versa, and it is applied with low probability. We introduce a specific mutation operator to generalize the attribute when almost every value is 1. In this case, the attribute does not appear in the rule. Mutation is always applied with probabilities 0.1 (individual) and 0.2 (gen).

C. Fitness Function

The fitness function must be able to discriminate between correct and incorrect example classifications. Finding an appropriate function, due to the noisy nature of most datasets, is not a trivial task. In our case, we try to both minimize the number of errors, and maximize the number of correctly classified examples. A simple solution to this two-objective optimization problem is considering both variables within the fitness function f, as shown in (1), where f is maximized for each individual φ from the population as follows:

$$f(\varphi) = 2(N - CE(\varphi)) + G(\varphi) + \text{coverage}\,(\varphi) \qquad (1)$$

where N is the number of examples being processed; $CE(\varphi)$ is the class error, which is produced when an example belongs to the region defined by the rule, but does not have the same class; $G(\varphi)$ is the number of examples correctly classified by the rule; and the *coverage* of a rule is the proportion of the search space covered by such rule. Each rule can be quickly expanded to find more examples, thanks to the coverage in the fitness function. The reason why $f(\varphi)$ is not $N - CE(\varphi) + G(\varphi) + \text{coverage}(\varphi)$ is as follows: for example, when $CE(\varphi) = 7$ and $G(\varphi) = 9$ we will have the same fitness value as when $CE(\varphi) = 15$ and $G(\varphi) = 17$ (the difference is 2; assuming the same coverage for both). Therefore, we decided to penalize the second case (9/7 is greater than 17/15) since less errors are preferred.

The coverage of a rule is calculated dividing the volume of the region defined by the rule by the whole volume of the search space. Let $[l_i, u_i]$ be the interval associated with an attribute i of the rule; k_i the number of active discrete values of an attribute i; $[L_i, U_i]$ the range of a continuous attribute i and $|A_i|$ the number of different values of a discrete attribute i. Then, the coverage of a rule φ is given by

$$\text{coverage}\,(\varphi) = \prod_{i=1}^{m} \frac{\text{coverage}\,(\varphi, i)}{\text{range}\,(\varphi, i)}$$

where

$$\text{coverage}\,(\varphi, i) = \begin{cases} u_i - l_i, & \text{if the attribute } i \text{ is continuous} \\ k_i, & \text{if it is discrete} \end{cases}$$

$$\text{range}\,(\varphi, i) = \begin{cases} U_i - L_i, & \text{if the attribute } i \text{ is continuous} \\ |A_i,| & \text{if it is discrete.} \end{cases}$$

IV. Results

The datasets used in this section are from the UCI Repository [31]. The results obtained by HIDER have been compared to that of C4.5 Release 8, C4.5Rules, See5, and See5Rules. To measure the performance of each method, a ten-fold cross-validation was achieved with each dataset (eighteen databases that involve continuous and/or discrete attributes). The algorithms were all run on the same training data and their induced knowledge structures tested using the same test data, so that the ten resulting performance numbers for C4.5Rules, C4.5, See5, See5Rules, and HIDER are comparable. It is very important to note that the experiments were run with the same default settings for all parameters of the EA: a population size of as little as 100 individuals and 300 generations. In cases of small datasets, like Iris, the results would have been the same using a smaller number of generations (about 50 had been enough). These are very small numbers, considering the number the examples and the dimensionality of some databases. HIDER needed about three hours to complete the ten-fold cross validation for the 18 databases in a Pentium 800 MHz with 256 Mb of RAM. C4.5 only needed about four minutes in the same machine and See5 about three minutes. Quinlan's tools are extremely robust algorithms that perform well in many domains. It is very difficult to consistently outperform them on a variety of datasets.

Table I gives the values of the parameters involved in the evolutionary process.

Table II gives the error rates (numbers of misclassified examples expressed as a percentage) for C4.5Rules, C4.5, See5, See5Rules, and HIDER algorithms on the selected domains. HIDER outperforms C4.5 and C4.5Rules in 12 out of 18 and 8 out of 18 datasets, respectively. If C4.5 produces bad trees, the results from C4.5Rules will not be very good. We can observe that there are four databases whose results generated by C4.5 are about 40% worse than those obtained by HIDER (breast cancer, iris, tic-tac-toe, and wine). It is especially worthy to note the

TABLE I
PARAMETERS OF HIDER

Parameter	Value
Population size	100
Generations	300
Crossover probability	0.5
Individual mutation probability	0.2
Gen mutation probability	0.1

TABLE II
COMPARING ERROR RATES

Database	See5Rules	See5	C4.5Rules	C4.5	HIDER
Breast Cancer (Wisconsin)	5.4	4.4	5.2	6.28	4.3
Bupa Liver Disorder	32.8	33.6	34.5	34.7	35.7
Cleveland	20.7	26.1	25.8	26.8	20.5
German Credit	28.0	28.4	28.8	32.1	29.1
Glass	36.5	33.2	18.5	32.7	29.4
Heart Desease	22.2	20.4	20.7	21.8	22.3
Hepatitis	18.0	20.0	16.9	21.4	19.4
Horse Colic	15.2	14.4	17.5	19.0	17.6
Iris	4.7	4.7	4.0	4.77	3.3
Lenses	10.0	16.7	16.7	29.9	25.0
Mushroom	0.2	0.0	0.0	0.0	0.8
Pima Indian	26.4	25.7	26.2	32.1	25.9
Sonar	26.5	33.2	29.3	30.3	43.1
Tic-Tac-Toe	3.7	14.9	18.8	14.2	3.8
Vehicle	27.5	29.9	57.6	30.6	30.6
Vote	4.4	28.5	5.3	6.2	6.4
Wine	7.3	6.2	6.7	6.7	3.9
Zoo	7.9	6.9	29.8	7.0	8.0
Average	16.5	19.3	20.1	19.8	18.3

error rate of the tic-tac-toe database. C4.5Rules improved the results of C4.5 for almost every database, except for three of them (tic-tac-toe, vehicle, and zoo). C4.5Rules did not achieve improvement of the results generated by C4.5, rather quite the opposite, it made the results worse, particularly for tic-tac-toe and zoo databases. See5Rules produced better results than HIDER for 12 databases, See5 for 10, C4.5Rules for 10 and C4.5 for 6. See5Rules obtained the best result for 7 databases, See5 for 5, C4.5Rules for 3, C4.5 for 1 and HIDER for 4. In general, See5 is more accurate than C4.5 and, above all, See5Rules reduced the error rate from 20.1 (C4.5Rules) to 16.5.

Table III compares the number of rules generated by the five approaches. In order to count the number of rules generated by C4.5, we can sum the leaves on the tree, or apply the expression $(s+1)/2$, where s is the size of the tree. C4.5Rules improves C4.5 in all databases, except mushrooms. These results are very similar to those generated by HIDER. For half of databases, less rules were generated with HIDER. Nevertheless, the result for german database is very interesting (5.2 rules), for others databases C4.5Rules reduces too much (3.3 rules for vehicle and 5.3 rules for zoo) which leads to a high error rate (57.6% for Vehicle and 29.8% for zoo). For this reason, although C4.5Rules on average generated less rules (7.9) than HIDER (9.5), the error

TABLE III
COMPARING NUMBER OF RULES

Database	See5Rules	See5	C4.5Rules	C4.5	HIDER
Breast Cancer (Wisconsin)	6.5	14.4	8.1	21.9	2.6
Bupa Liver Disorder	16.3	29.3	14.0	28.6	11.3
Cleveland	13.5	22.7	11.3	35.2	7.9
German Credit	39.0	71.2	5.2	181.5	13.3
Glass	16.1	23.8	14.0	29.0	19.0
Heart Desease	10.7	17.9	10.5	29.2	9.2
Hepatitis	6.9	9.6	5.4	13.8	4.5
Horse Colic	4.4	5.8	4.1	39.3	6.0
Iris	4.2	4.7	4.0	5.5	4.8
Lenses	3.7	3.7	3.1	4.1	6.5
Mushroom	9.2	19.0	17.2	15.7	3.1
Pima Indian	15.1	28.8	9.8	93.6	16.6
Sonar	15.5	15.1	5.1	16.8	2.8
Tic-Tac-Toe	21.9	86.8	10.7	93.9	11.9
Vehicle	52.9	67.1	3.3	102.3	36.2
Vote	5.4	72.2	6.6	14.7	4.0
Wine	4.8	5.4	4.6	5.4	3.3
Zoo	8.3	8.4	5.3	9.9	7.2
Average	14.1	28.1	7.9	41.1	9.5

rate increased considerably, C4.5Rules (20.1%) versus HIDER (18.3%). See5Rules produced better results than HIDER for five databases, See5 for three, C4.5Rules for nine and C4.5 for one. The best results were generated by C4.5Rules (9 databases) and HIDER (9 databases).

Table IV shows a measure of improvement (ϵ) for the error rate [first column: (ϵ_{er})] and the number of rules [second column: (ϵ_{nr})] used in the Quinlan's works [32]. To calculate these coefficients, ϵ_{er} and ϵ_{nr}, respectively, the error rate (number of rules) for each method has been divided by the corresponding error rate (number of rules) for HIDER. On average, HIDER found solutions that had less than one forth of the rules output by C4.5. Surprisingly, C4.5 generated a number of rules 5 times greater than HIDER for one third of the databases. It is worth noting that in applying HIDER, more than two-thirds of the databases produce less than half the rules. C4.5 only was better with Lenses database. C4.5 made the error rate better for six databases, although only three of them improved significantly (mushrooms, sonar, and zoo). In summary, the average error rate generated by C4.5 is 22% greater and the average number of rules 340%. This reason leads us to compare our approach with C4.5Rules and See5Rules, mainly in regard to the number of rules. The average ratio of the error rate of C4.5 to that of HIDER is 1.22, while the ratio of the number of rules is 4.40.

Although the results in Table III indicated that C4.5Rules improved on average (7.9 rules) to HIDER (9.5 rules), analyzing the relative increase of the number of rules we can observe that those numbers can be deceptive. C4.5Rules generates an average number of rules 33% greater than HIDER, as well as an average error rate 36% higher, as shown in the last row of Table IV.

In general, taking into account both the error rate and the number of rules, See5Rules improves HIDER for one database;

TABLE IV
COMPARING GLOBAL RESULTS

Database	$\frac{See5Rules}{HIDER}$		$\frac{See5}{HIDER}$		$\frac{C4.5}{HIDER}$		$\frac{C4.5Rules}{HIDER}$	
	ϵ_{er}	ϵ_{nr}	ϵ_{er}	ϵ_{nr}	ϵ_{er}	ϵ_{nr}	ϵ_{er}	ϵ_{nr}
Breast Cancer (Wisconsin)	1.26	2.50	1.03	5.54	1.46	8.42	1.21	3.12
Bupa Liver Disorder	.92	1.44	.94	2.59	.97	2.53	.97	1.24
Cleveland	1.01	1.71	1.27	2.87	1.31	4.46	1.26	1.43
German Credit	.96	2.93	.98	5.35	1.10	13.65	.99	.39
Glass	1.24	.85	1.13	1.25	1.11	1.53	.63	.74
Heart Desease	.99	1.16	.91	1.95	.98	3.17	.93	1.14
Hepatitis	.93	1.53	1.03	2.13	1.10	3.07	.87	1.20
Horse Colic	.86	.73	.82	.97	1.08	6.55	.99	.68
Iris	1.41	.88	1.41	.98	1.40	1.15	1.21	.83
Lenses	.40	.57	.67	.57	1.20	.63	.67	.48
Mushroom	.26	2.97	.00	6.13	.01	5.00	.01	5.55
Pima Indian	1.02	.91	.99	1.73	1.24	5.64	1.01	.59
Sonar	.62	5.54	.77	5.39	.70	6.00	.68	1.82
Tic-Tac-Toe	.96	1.84	3.87	7.29	3.69	7.89	4.95	.90
Vehicle	.90	1.46	.98	1.85	1.00	2.83	1.88	.09
Vote	.69	1.35	4.44	18.05	.96	3.68	.83	1.65
Wine	1.85	1.45	1.57	1.64	1.70	1.64	1.72	1.39
Zoo	.99	1.15	.86	1.17	.88	1.38	3.72	.74
Average	.96	1.72	1.31	3.75	1.22	4.40	1.36	1.33

TABLE V
NUMBER OF DATABASES FOR WHICH THE METHOD IMPROVES HIDER

Better than HIDER	Error Rate	Number of Rules	Both
See5	10	3	2
See5Rules	12	5	1
C4.5	6	1	0
C4.5Rules	10	9	4

See5 for two; and C4.5Rules for four (C4.5 for none). Although See5 behaved well with respect to the error rate, it was not as good as expected, since when the error rate was improved, the number of rules increased, and vice versa. C4.5Rules behaved better on average from this point of view. The most accurate out of Quinlan's tools is See5Rules, although it generates more rules than C4.5Rules.

Table V gives an idea of the performance of each method in comparison to HIDER, with respect to the number of rules and/or the error rate. No one tool is significantly better than HIDER, so it is a robust approach.

V. CONCLUSIONS

An EA-based supervised learning tool to classify databases is presented in this paper. HIDER produces a hierarchical set of rules, where each rule is tried in order until one is found whose conditions are satisfied by the example being classified. The use of hierarchical decision rules lead to overall improvement of the performance on the eighteen databases investigated here. In addition, HIDER improves the flexibility to construct a classifier varying the relaxing coefficient. In other words, one can trade off accuracy against understanding. HIDER was compared to C4.5, C4.5Rules, See5, and See5Rules and both the number of rules and error rate were decreased. Out of Quinlan's tools, we can state on basis of experiments that See5Rules has the best average performance as for error rate as number of rules. To summarize shortly, the experiments show that HIDER works well in practice.

VI. FUTURE WORKS

Evolutionary algorithms are very time consuming. This aspect is being analyzed from the point of view of the coding. Another aspect being studied is the way in which the evaluation function analyzes each example from the database. Research on improvements to data structure as input of EAs, in order to reduce the time complexity, is currently being conducted.

ACKNOWLEDGMENT

Author J. S. Aguilar-Ruiz is grateful to R. López de Mántaras for the constructive criticisms and for evaluating his doctoral dissertation. The authors would like to thank the reviewers for their comments and interesting suggestions.

REFERENCES

[1] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[2] S. K. Murthy, S. Kasif, and S. Salzberg, "A system for induction of oblique decision trees," *J. Artif. Intell. Res.*, 1994.

[3] R. L. Rivest, "Learning decision lists," *Machine Learning*, vol. 1, no. 2, pp. 229–246, 1987.
[4] P. Clark and T. Niblett, "The cn2 induction algorithm," *Mach. Learn.*, vol. 3, no. 4, pp. 261–283, 1989.
[5] P. Clark and R. Boswell, "Rule induction with cn2: Some recents improvements," in *Machine Learning: Proc. 5th Eur. Conf.*, 1991, pp. 151–163.
[6] "Proc. American Association Artificial Intelligence Conf.," Univ. Illinois, IL, Tech. Rep. UIUCDCS-R-86–1260, 1986.
[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[8] D. Whitley, "A genetic algorithm tutorial," Colorado State Univ., Ft. Collins, CO, Tech. Rep. CS-93-103, 1993.
[9] S. Forrest, "Genetic algorithms," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 77–80, 1996.
[10] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. New York: Springer-Verlag, 1996.
[11] 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.
[12] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Mach. Learn.*, vol. 1, no. 13, pp. 169–228, 1993.
[13] G. Venturini, "SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts," in *Proc. Eur. Conf. Machine Learning*, 1993, pp. 281–296.
[14] E. Cantu-Paz and C. Kamath, "Using evolutionary algorithms to induce oblique decision trees," in *Proc. Genetic Evolutionary Computation Conf.*, Las Vegas, NV, 2000, pp. 1053–1060.
[15] D. R. Carvalho and A. A. Freitas, "A hybrid decision tree/genetic algorithm for coping with the problem of small disjuncts in data mining," in *Proc. Genetic Evolutionary Computation Conf.* , Las Vegas, NV, 2000, pp. 1061–1068.
[16] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "An evolutionary approach to estimating software development projects," *Inf. Softw. Technol.*, vol. 14, no. 43, pp. 875–882, 2001.
[17] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Foundations of Genetic Algorithms-2*. San Mateo, CA: Morgan Kaufman, 1993, pp. 187–202.
[18] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, *A Tool to Obtain a Hierarchical Qualitative Set of Rules From Quantitative Data*. New York: Springer-Verlag, 1998, pp. 336–346.
[19] 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.
[20] S. Bhattacharyya and G. J. Koehler, "An analysis of nonbinary genetic algorithms with cardinality 2^v," *Complex Syst.*, vol. 8, pp. 227–256, 1994.
[21] G. J. Koehler, S. Bhattacharyya, and M. D. Vose, "General cardinality genetic algorithms," *Evol. Computation*, vol. 5, no. 4, pp. 439–459, 1998.
[22] M. D. Vose and A. H. Wright, "The simple genetic algorithm and the walsh transform: Part I, Theory," *Evol. Computation*, vol. 6, no. 3, pp. 253–273, 1998.
[23] ——, "The simple genetic algorithm and the walsh transform: Part II, The inverse," *Evol. Computation*, vol. 6, no. 3, pp. 275–289, 1998.
[24] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "Three geometric approaches for representing decision rules in a supervised learning system," in *Proc. Genetic Evolutionary Computation Conf.* , vol. EE.UU., Orlando, FL, 1999, p. 771.
[25] T. Mitchell, *Machine Learning*. New York: McGraw Hill, 1997, ch. 10.
[26] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Mach. Learn.*, vol. 11, pp. 63–91, 1993.
[27] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of Genetic Algorithms-1*. San Mateo, CA: Morgan Kaufman, 1991, pp. 205–218.
[28] N. J. Radcliffe, "Genetic neural networks on MIMD computers," Ph.D. dissertation, Univ. Edinburgh, Edinburgh, U.K., 1990.
[29] G. Syswerda, "Uniform crossover in genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 2–9.
[30] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *J. Artif. Intell. Res.*, vol. 6, no. 1, pp. 1–34, 1997.
[31] C. Blake and E. K. Merz, *UCI Repository of Machine Learning Databases*, 1998.
[32] J. R. Quinlan, "Improved use of continuous attributes in c4.5," *J. Artif. Intell. Res.*, vol. 4, pp. 77–90, 1996.

Jesús S. Aguilar-Ruiz received the B.Sc. degree in 1983, the M.Sc. degree in 1992, and the Ph.D. degree in 1996, all in computer science, from the University of Seville, Seville, Spain.

He is an Associate Professor at the School of Computer Science at the University of Seville, Spain. He has been member of the Programm Committee of several international conferences, as ACM Knowledge Discovery and Data Mining Conference (KDD) or Genetic and Evolutionary Computation Conference (GECCO). His areas of research interest include evolutionary algorithms, knowledge discovery, data mining and bioinformatics.

He received his doctoral dissertation award from the University of Seville. http://www.lsi.us.es/~aguilar.

José C. Riquelme received the B.Sc. degree in 1983, the M.Sc. degree in math, in 1985, and the Ph.D. degree in 2001 in computer science, all from the University of Seville, Seville, Spain.

He is Associate Professor at the University of Seville, Spain. He is an expert in genetic programming. 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. He has also edited some proceedings. http://www.lsi.us.es/~riquelme.

He received his doctoral dissertation award from the University of Seville.

Miguel Toro (M'95) received the B.Eng. degree in 1980, the M.S.Eng. in 1982, and the Ph.D. degree in 1987 in dynamical systems, from the Univeristy of Seville, Seville, Spain.

He is a Professor at the University of Seville, Spain. He is author of several books on dynamical systems. He is a recognized researcher in Software Engineering and leads one of the most important groups in Spain. http://www.lsi.us.es/~mtoro.