# Handling Continuous Attributes in an Evolutionary Inductive Learner

Federico Divina and Elena Marchiori

*Abstract*—**This paper analyzes experimentally discretization algorithms for handling continuous attributes in evolutionary learning. We consider a learning system that induces a set of rules in a fragment of first-order logic (evolutionary inductive logic programming), and introduce a method where a given discretization algorithm is used to generate initial inequalities, which describe subranges of attributes' values. Mutation operators exploiting information on the class label of the examples (supervised discretization) are used during the learning process for refining inequalities. The evolutionary learning system is used as a platform for testing experimentally four algorithms: two variants of the proposed method, a popular supervised discretization algorithm applied prior to induction, and a discretization method which does not use information on the class labels of the examples (unsupervised discretization). Results of experiments conducted on artificial and real life datasets suggest that the proposed method provides an effective and robust technique for handling continuous attributes by means of inequalities.**

*Index Terms*—**Discretization, evolutionary computation, inductive concept learning (ICL).**

## I. Introduction

THE TASK OF learning a target concept in a given representation language, from a set of positive and negative realizations of that concept (examples) and some background knowledge, is called inductive concept learning (ICL) [1]. If the representation language is a fragment of first-order logic then it is called inductive logic programming (ILP) [2].

Real-life learning tasks are often described by nominal as well as continuous, real-valued, attributes. However, most integer linear programming (ILP) systems treat all attributes as nominal. Hence, such systems cannot exploit the linear order of real values, if real values are treated as nominal ones. This limitation may have a negative effect not only on the execution speed but also on the learning capabilities of such systems.

In order to overcome these problems, more involved transformations of continuous-valued attributes into nominal ones are applied. For instance, the range of the attribute's values are split in a finite number of intervals, which are treated as values of a nominal attribute. Alternatively, continuous attributes are handled by means of inequalities describing attribute subranges, whose boundaries are computed during the learning process. This process, called discretization, is supervised when it uses the class labels of examples, and unsupervised, otherwise. Discretization can be applied prior to or during the learning process (global and local discretization, respectively), and can either discretize one attribute at a time (univariate discretization) or take into account attributes interdependencies (multivariate discretization) [3].

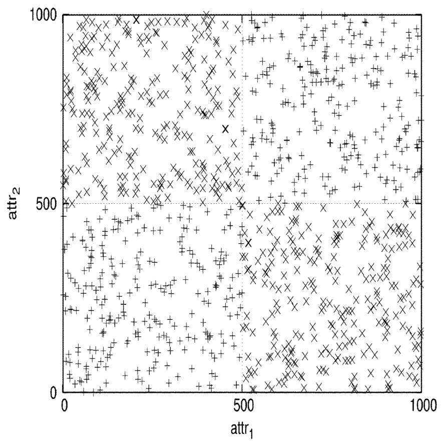


Fig. 1. A problem for which univariate discretization is unlikely to work.

Researchers in the machine learning community have introduced many discretization algorithms. An overview of various types of discretization algorithms can be found, e.g., in [4]–[7]. Most of these algorithms perform an iterative greedy heuristic search in the space of candidate discretizations, using different types of scoring functions for evaluating a discretization. For instance, the popular Fayyad and Irani discretization algorithm [8] considers one attribute at a time, uses an information class entropy measure for choosing a cut point yielding a partition of the attribute domain, applies recursively the procedure to both the partitions, and uses the minimum description length as criterion for stopping the recursion.

A typical example showing a drawback of univariate discretization methods based on class information entropy is the problem of separating the two classes shown in the Fig. 1, where positive and negative examples are labeled $+$ and $\times$.

Any cut point divides the domain of one attribute in two partitions having approximately the same class distribution as the entire domain. Thus, a condition on a single attribute does not improve class separation, so univariate supervised discretization methods based on information class entropy are unlikely to work.

An elegant and robust approach for overcoming this drawback is provided by evolutionary algorithms, which can be used

Manuscript received September 23, 2003; revised March 3, 2004.

The authors are with the Department of Computer Science, Vrije Universiteit van Amsterdam, 1081 HV Amsterdam, The Netherlands (e-mail: divina@cs.vu.nl; elena@cs.vu.nl).

Digital Object Identifier 10.1109/TEVC.2004.837752

for performing local multivariate discretization during the evolutionary learning process. Recently, several methods have been introduced based on this approach. In these methods numerical values are handled by means of inequalities that can be modified, by means of ad hoc operators, during the evolutionary process. These methods differ among each other mainly in the way inequalities, describing continuous attribute subranges, are encoded, and in the definition of suitable genetic operators for modifying inequalities. An overview of some of these methods is given in Section V.

We will use evolutionary learners based on a more expressive representation, like the ILP system, in this paper. Called evolutionary concept learning (ECL) [9], it generally treats continuous attributes as nominal ones or discretize them prior to induction, e.g., using Fayyad and Irani algorithm. Recently [10] proposed an unsupervised local multivariate discretization method which is embedded in ECL. The resulting system, here called ECL-LUD (ECL with local unsupervised discretization), evolves rules containing inequalities. An inequality is introduced in a rule each time a continuous attribute value of an example is considered, and the inequality boundaries are initialized to that value. Then, during evolution, mutation operators using information about the density of the values of an attribute, are applied for shifting the boundaries of the inequalities. It is shown that this unsupervised local discretization method improves the performance of ECL on a number of classification tasks.

In this paper, we analyze experimentally the effect of other discretization methods for ECL, in particular, local supervised multivariate discretization and test, and compare experimentally the resulting variants of ECL with other classification methods.

We propose a discretization method that uses the intervals generated by a given (global supervised univariate discretization) algorithm for initializing the inequalities introduced in a rule, and refines these inequalities, during the evolutionary process, by means of mutation operators, which use specific cut points for shifting inequality boundaries. More specifically, we consider the two following possible initializations of inequalities: a fine-grain initialization, using intervals formed by two consecutive *boundary points*, where a boundary point is the midpoint of two consecutive attribute values having different class labels [11], and a coarser grain initialization, using intervals obtained from the Fayyad and Irani algorithm (outlined above). During the evolutionary process the mutation operators use the boundary points for modifying the inequalities. The resulting ECL variants are called ECL-LSDf and ECL-LSDc, respectively.

We compare experimentally four variants of ECL with discretization: ECL with global univariate discretization (ECL-GSD), which uses Fayyad and Irani algorithm prior to evolution, ECL with local multivariate unsupervised discretization (ECL-LUD), and ECL with the two variants (ECL-LSDf and ECL-LSDc) of the proposed local supervised discretization method.

We analyze experimentally the performance of the four ECL variants on the nonlinearly separable problem described above and on real-life propositional and relational datasets. On the real-life datasets, ECL-LSDc is the best performing system. However, as expected, it is unable to solve the nonlinearly separable problem, while ECL-LSDf is able to solve this problem, but its fine-grain initialization of inequalities sometimes leads the system to overfit the training data.

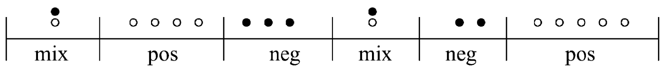


Fig. 2. Example of boundary points of an attribute: ∘ denotes a value occurring in a positive example, while • a value occurring in a negative one.

In general, the results of the experiments indicate that initializing inequalities using intervals obtained from Fayyad and Irani algorithm [11] and then refining them during the learning process in order to take into account the possible attribute interdependencies, provides a robust and effective technique for handling continuous attributes in evolutionary ILP learning.

The rest of the paper is organized as follows. In Section II, we describe the two types of cut points used as boundaries of inequalities and the operators for shifting inequalities boundaries employed in the mutation. Next, we briefly overview the ECL system and its four extensions with discretization. In Section IV, we report and discuss the results of experiments, and we compare the best results obtained by the various settings of ECL with results obtained by other ICL methods. In Section V, we consider related work, and finally, in Section VI, we give some conclusions.

## II. Generating Boundary Points for Inequalities

The discretization method we propose uses the following two types of cut points, called boundary and discretization points.

### A. Boundary Points

Boundary points have been introduced and analyzed in [11]. Given a numeric attribute $A$ and a set of positive and negative examples, the values of $A$ occurring in the examples are sorted in increasing order. A *boundary point* is the midpoint of two successive values of $A$ occurring in examples of different classes. Here, we call boundary points also the smallest and biggest value of $A$, denoted by $-\infty$ and $\infty$, respectively.

Each pair of consecutive boundary points describes an interval, which can be of three types: *negative* if its values occur only in negative examples, *positive* if they occur only in positive examples, and *mixed* if the interval contains just one value, and this value occurs both in a positive and a negative example. An example is shown in Fig. 2.

We denote by $\mathrm{BP}(A)$ the sequence of boundary points of $A$ sorted in increasing order, and call BP *interval* an interval defined by two successive elements of $\mathrm{BP}(A)$. Boundary points are sufficient for finding the minimum of class information entropy, a measure used in the following discretization algorithm [11].

### B. Discretization Points

Fayyad and Irani discretization algorithm uses the class information entropy of candidate intervals to select the boundaries of the intervals for discretization.

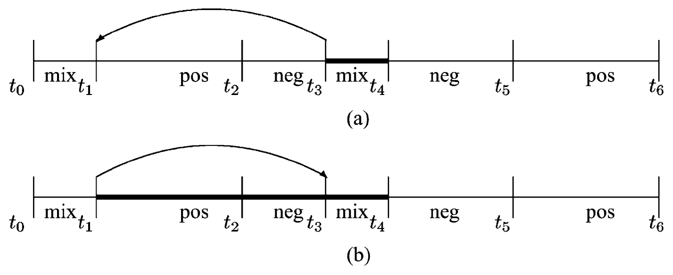


Fig. 3. (a) Application of enlarge. (b) Application of shrink. The inequality is represented by a thick segment.

Given a set $S$ of instances, an attribute $A$, and a threshold point $t$, the class information entropy of the partition induced by $t$ is given by

$$E(A,t,S) = \text{Entropy}(S_1)\frac{|S_1|}{|S|} + \text{Entropy}(S_2)\frac{|S_2|}{|S|}$$

where $S_1$ and $S_2$ are the sets of instances whose values of $A$ are in the first and second half of the partition. Moreover, $|S|$ denotes the number of elements of $S$ and $\text{Entropy}(S) = -A_+ \log_2(A_+) - A_- \log_2(A_-)$ with $A_+$ and $A_-$ the proportions of positive and negative examples of $S$.

The algorithm searches for a boundary point $t^*$ which minimizes $E(A,t,S)$. Such $t^*$, here called *discretization point*, is selected as boundary of a binary discretization. The method is applied recursively to both the partitions induced by $t^*$ until a stopping criterion is satisfied. The minimum description length principle [12] is used in the stopping criterion. The recursive process within a set of instances stops if $\text{Entropy}(S) - E(A,t,S)$ is smaller than $\log_2(N-1)/N + \Delta(A,t,S)/N$, where $\Delta(A,t,S) = \log_2(3^k - 2) - [k \cdot \text{Entropy}(S) - k_1 \cdot \text{Entropy}(S_1) - k_2 \cdot \text{Entropy}(S_2)]$, and $k_i$ is the number of class labels represented in $S_i$.

In the discretization method of Fayyad and Irani, the intervals of the final partition are treated as values of a nominal attribute.

We denote by $\text{DP}(A)$ the sequence of discretization points sorted in increasing order, and call DP *interval* an interval defined by two consecutive elements of $\text{DP}(A)$.

## C. Enlarging and Shrinking Inequalities

We handle a numeric attribute $A$ by means of inequalities of the form $l < A \leq u$, where $l < u$ are specific elements of $\text{BP}(A)$. An element of $\text{BP}(A)$ is called *left-good* if it is not the left boundary of a negative $\text{BP}(A)$ interval, and *right-good* if it is not the right boundary of a negative $\text{BP}(A)$ interval. We will consider only inequalities $l < A \leq u$ with $l$ and $u$ left- and right-good, describing intervals that do not start or end with a negative $\text{BP}(A)$ interval.

For instance, assume the boundary points of $A$ are those in Fig. 3. Then, $t_0 < A \leq t_2$ is a legal inequality, while $t_1 < A \leq t_3$ is not legal, because it describes an interval that ends with a negative $\text{BP}(A)$ interval.

Now, assume $\text{BP}(A) = (t_0, \ldots, t_n)$, and consider the inequality $t_i < A \leq t_j$ with $i,j \in [0,n], i < j$ and $t_i, t_j$ left- and right-good. We introduce the following generalization and specialization operators.

`enlarge`:

1) Randomly select either $t_i$ or $t_j$.
2) a) If $t_i$ has been chosen, find the greatest $t_{i'}$ such that $i' < i$ and $t_{i'}$ is left-good. Set $t_i$ to $t_{i'}$. If such $t_{i'}$ does not exist (if $i = 0$ or all intervals to the left of $t_i$ are negative), then go to step *b)* if it was not already tried.
   b) If $t_j$ has been chosen, find the smallest $t_{j'}$ such that $j' > j$ and $t_{j'}$ is right-good. Set $t_j$ to $t_{j'}$. If such $t_{j'}$ does not exist (if $j = n$ or all intervals to the right of $t_j$ are negative), then go to step *a)* if it was not already tried.

`shrink`:

This operator is applicable if $|i - j| > 1$.

1) Randomly select either $t_i$ or $t_j$.
2) a) If $t_i$ has been chosen, find the smallest $t_{i'}$ such that $i' < j, i' > i$ and $t_{i'}$ is left-good. Set $t_i$ to $t_{i'}$.
   b) If $t_j$ has been chosen, find the greatest $t_{j'}$, where $j' < j, j' > i$ and $t_{j'}$ is right-good. Set $t_j$ to $t_{j'}$.

Notice that application of enlarge and shrink preserves the left- and right-goodness of the boundaries of an inequality.

Fig. 3 illustrates the application of the enlarge and shrink operators to inequalities represented by the thick lines. Enlarge applied to $t_3 < A \leq t_4$ shifts its left boundary $t_3$ to $t_1$, while shrink applied to $t_1 < A \leq t_4$ shifts its left boundary $t_1$ to $t_3$.

## III. ECL Plus Discretization

Evolutionary concept learning (ECL) is an evolutionary ILP learner, which takes as input a background knowledge here and in the sequel denoted by BK, a set of positive and negative examples of the target concept, and outputs a set of rules in a fragment of first-order logic, called clauses, that covers many positive examples and few negative ones.

The main features of ECL are: 1) a high-level encoding of clauses which allows the direct application of standard ILP generalization and specialization operators; 2) a random sampling mechanism for selecting a portion of the background knowledge which improves efficiency; 3) greedy mutation operators for guiding the search; and 4) a simple optimization procedure applied to each individual after mutation. The system induces an approximated model of the target concept.

Here, and in the sequel, we use `Prolog` convention where variables and constants are denoted by words starting with a capital and a lowercase letter, respectively.

ECL evolves rules of the form

$$p(X,Y) : -r(X,Y,Z), q(Z,a)$$

where $p(X,Y), r(X,Y,Z), q(Z,a)$ are atoms, consisting of a predicate symbol and a number of arguments which may be either variables or constants, $p(X,Y)$ is called the head and $r(X,Y,Z), q(Z,a)$ the body of the clause. A clause has the declarative interpretation in first-order logic

$$\forall X,Y,Z(r(X,Y,Z), q(Z,a) \rightarrow p(X,Y))$$

and the procedural one

in order to solve $p(X,Y)$ solve $r(X,Y,Z)$ and $q(Z,a)$

```
ALGORITHM ECL
Sel = positive_examples
repeat
    Select partial BK
    Population = ∅
    while (not terminate) do
        Select sel chromosomes
        for each selected chromosome chrm
            Mutate chrm
            Optimize chrm
            Insert chrm in Population
        end for
    end while
    Store Population in Final_Population
until max_iter is reached
Extract final rules from Final_Population
```

Fig. 4. Overall learning algorithm ECL.

where each atom is viewed as a procedure call. A set of clauses is called logic program and can directly be executed in the programming language Prolog.

The background knowledge used by ECL consists of clauses with empty body containing only constants as arguments, called ground facts. A clause *covers an example* if the theory formed by the clause and the background knowledge logically entails the example.

A schematic illustration of ECL is given in pseudo-code in Fig. 4. A main loop is used for constructing incrementally a Final_population as the union of max_iter populations computed at each iteration of the repeat statement.

At each iteration part of the BK is randomly sampled, using a user defined parameter *pbk* which specifies the probability of selection of a BK fact. This sampling is performed in order to reduce the computational effort required by the evaluation of individuals. By using a *pbk* smaller than one, part of the background knowledge is not used in the evolutionary learning process. This speeds up the evaluation of individuals but renders the evaluation "incomplete."

The resulting partial BK, and the examples that it covers, are fed to an evolutionary algorithm (the while statement) that induces a Population of clauses.

The fitness of a clause is equal to its accuracy, which is the number of examples correctly classified by the clause divided by the total number of examples. In order to establish the number of examples correctly classified by a clause, Prolog is run with the clause and the background knowledge as program and each example as query.

The evolutionary process searches for clauses with the maximal fitness. The algorithm starts from an empty population and evolves clauses using selection, mutation, and optimization. ECL does not use any crossover operator. The reason behind this choice is that it is difficult to design an effective crossover operator with the high level representation adopted by ECL. Some experiments were conducted with a uniform crossover, but the results of such experiments did not justify its use.

At each generation, a number specified by a user defined parameter *sel*—of offspring is generated and inserted in the actual population as follows. An individual of the population is chosen using a slight modification, introduced in [13] and extensively validated in [14], of the so-called universal suffrage (US) selection operator [15]. This operator works in two steps: first, a positive example is selected by a mechanism that favors "harder" examples, that is, covered by few clauses. In order to determine the "hardness" of an example, a weight is assigned to each example, and is adjourned at every generation. The weight depends on the number of individuals covering the example. This is different from the standard US selection operator, where examples are randomly chosen. Next, a roulette wheel is performed on the individuals of the actual Population covering that example. If the selected example is not covered by any individual (for instance when the population is empty) then a new clause is created as follows. The example becomes the head of the clause, and suitable elements of the (partial) background knowledge BK having arguments in common with those of the example are added to the body of the emerging clause. As in most ILP systems, a maximum number of body atoms is allowed, which is specified by a user defined parameter $lc$. All individuals try to predict the same class in the same run. This means that the head of the clause contains the same predicate symbol for all the individuals throughout a run. This implies that when a $K$-class, $K > 2$, problem is tackled, ECL is run $K$ times, once per each class. The $K$ learned theories are then combined as described in Section IV-D.

Mutation uses standard ILP generalization and specialization operators. A clause is generalized using either the "delete an atom" operator which removes an atom from its body, or the "constant into variable" operator which turns a constant into a variable. Dually, a clause is specialized using either the "add an atom" operator which adds an atom to the body of the clause, or the "variable into constant" operator which turns a variable into a constant. The choice of which operator to apply is random. The user can tune the greediness of each operator by means of a parameter $N_i, i \in [1, 4]$ ($N_1$ is associated to "delete an atom," $N_2$ to "constant into variable," $N_3$ to "add an atom," and $N_4$ to "variable into constant"). Each $N_i$ specifies the number of clauses that are generated by applying the corresponding operator. The best (in terms of fitness) of the $N_i$ generated clauses is chosen as offspring.

Optimization consists of the repeated application of generalization and specialization operators, while the fitness remains equal or improves and a given maximum number of applications is not reached.

Each mutated and optimized individual is inserted in the population as follows: if the actual population has not yet reached its maximum size then the new individual is just added, otherwise, the new individual replaces an individual of the actual population chosen using four—tournament selection. The tournament size was experimentally determined.

At the end of the repeat statement, after max_iter evolutionary algorithms have been executed, using possibly different portions of BK, and the resulting populations have been joined in Final_Population, the system extracts a final set of clauses—a Prolog program—from Final_Population. Such a set is incrementally constructed from the empty set as follows [16]. A clause with maximum *precision*[1] is moved from `Final_Population` to the actual final set of clauses (in case of ties, the clause covering more positive examples is chosen), the examples covered by this clause are discarded, and the precision of the remaining clauses in `Final_Population` is recomputed. The process is iterated, while the accuracy of the actual final set of clauses does not decrease.

In [10], another method is used for extracting the final solution, which uses only information about the accuracy of a clause and produces results of inferior quality.

## A. *Four Variants of ECL for Discretization*

We have embedded four discretization methods in ECL. The user can select the preferred method by setting a corresponding parameter when running the system.

1) ECL-GSD, where Fayyad and Irani discretization algorithm (described in Section II-B) is applied prior to induction.
2) ECL-LUD [10], with local unsupervised discretization, described in Section I.
3) ECL-LSDc, with local supervised discretization and a coarse initialization of inequalities using DP points.
4) ECL-LSDf, as the previous variant but with a fine initialization of inequalities using BP points.

The last three methods use inequalities, which are introduced in a clause when an atom containing a numeric value is added to its body. This happens when the clause is generated or during clause evolution, as illustrated in the following example.

Assume the clause

$$\text{target}(c) : -q(c,a), t(c,b)$$

is constructed with the example $\text{target}(c)$ as seed. If the specialization operator "add an atom" is chosen and the BK fact $r(c, 8.23)$ is selected, then the clause becomes

$$\text{target}(c) : -q(c,a), t(c,b), r(c,X), \quad 8.23 \leq X \leq 8.23.$$

in ECL-LUD

$$\text{target}(c) : -q(c,a), t(c,b), r(c,X), \quad l < X \leq u$$

in ECL-LSDf, where $l, u$ are boundaries of the BP interval containing 8.23, and

$$\text{target}(c) : -q(c,a), t(c,b), r(c,X), \quad l < X \leq u$$

in ECL-LSDc, where $l, u$ are boundaries of the DP interval containing 8.23.

The same operators are used in ECL-LSDf and ECL-LSDc for evolving rules. If the generalization operator "delete an atom" is chosen and $r(c,X)$ is selected for deletion, then $r(c,X)$ and the corresponding inequality are removed from the clause. The other possible generalization operator consists of a random choice between the "constant into variable," which replaces one of the constants $a, b, c$ with a variable, and the "enlarge" operator, which enlarges one boundary of the inequality. If the specialization operator "variable into constant" is chosen and the variable $X$ is selected, then the "shrink" operator is applied to the relative inequality.

In ECL-LUD suitable operators defined on an inequality modify its boundaries using information, on the distribution of the values of the corresponding attribute, obtained by clustering the values with the expectation-maximization algorithm [17].

[1]Measures often used for evaluating a clause are recall $= \text{TP}/(\text{TP} + \text{FN})$, precision $= \text{TP}/(\text{FP} + \text{TP})$, and accuracy $= (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN})$, where TP and TN are the number of positive and negative examples correctly classified, and FP and FN are the number of negative and positive examples wrongly classified.

## IV. EXPERIMENTS

First, we use the artificial dataset discussed in the Introduction for analyzing the behavior of the four ECL variants on this nonlinearly separable problem. Next, we consider real-life learning tasks and perform experiments on propositional and relational datasets, chosen for the high presence of numeric attributes. After this, the results obtained by ECL are compared with those obtained by other systems for ICL.

## A. *Artificially Generated Dataset*

In this experiment, 50 positive and 50 negative examples of the target concept described in Section I are fed to the system. So examples of the positive class are realizations of the target concept. Each example is described by two attributes that can take values in $[0, 1000]$. The system is run with population size equal to 100, for 50 generations and with 30 individuals selected at each generation, while the greediness $N_i$ of the mutation operators and the maximum length of a clause $lc$ are set to 3. All the background knowledge is used, thus, $pbk$ is set to 1, and a maximum of ten optimization steps is performed. ECL is used for inducing rules predicting the positive examples.

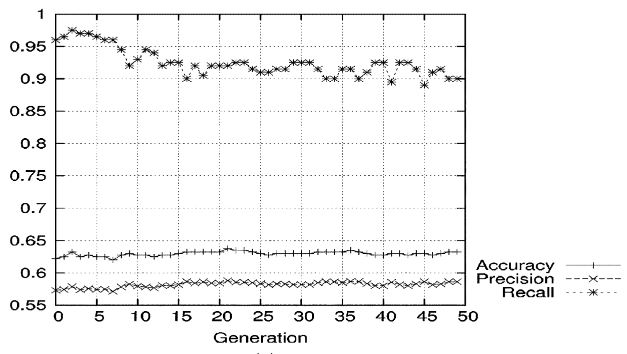
ECL-GSD is not able to solve this problem because no discretization point is found by the Fayyad and Irani method. For the same reason, ECL-LSDc has scarce performance, as shown in Fig. 5(a), where the average accuracy, precision, and recall over five runs of the extracted solution are plotted at each generation, indicating that there is no evolution.

The other two ECL variants, ECL-LUD and ECL-LSDf, have satisfactory performance. Fig. 5(b) shows the average results of five runs of ECL-LUD, where the accuracy of the extracted solution can be seen to increase, even if rather slowly, during the generations, and it becomes practically constant after about 40 generations, where all the properties become about 0.97.

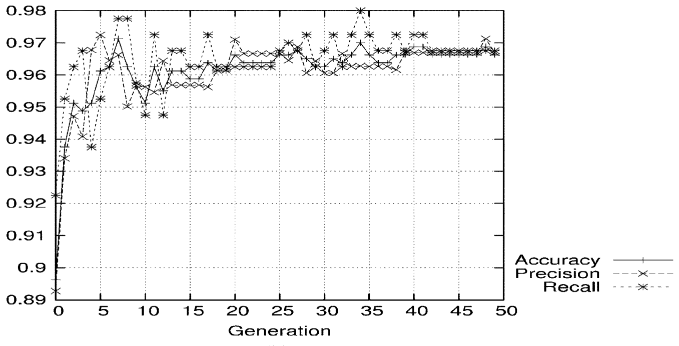
Fig. 6 shows the average accuracy, precision, and recall of the solution extracted at each of the 50 generations of five runs of ECL-LSDf. It can be seen that after some oscillations a perfect solution is found, like the one consisting of the following two clauses.

$$\begin{aligned}&\mathtt{case(X)} : -\mathtt{attr_1(X,Y)}, \mathtt{attr_2(X,Z)}, (611.56 < \mathtt{Z} \leq 976.55),\\ &\qquad (516.79 < \mathtt{Y} \leq 975.36).\\ &\mathtt{case(X)} : -\mathtt{attr_1(X,Y)}, \mathtt{attr_2(X,Z)}, (-\infty < \mathtt{Z} \leq 487.58),\\ &\qquad (21.24 < \mathtt{Y} \leq 489.59).\end{aligned}$$

Fig. 7(a) shows the average number of positive and negative examples covered by the individuals of the population at every



(a)



(b)

Fig. 5. Average accuracy, precision, and recall of the solution found at every generation of five runs of ECL-LSDc and ECL-LUD. (a) ECL-LSDc. (b) ECL-LUD.

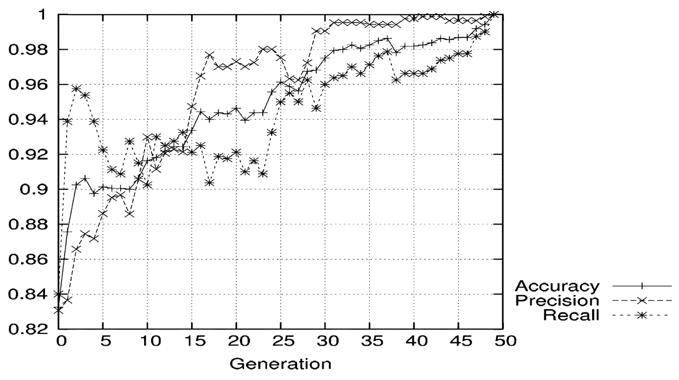


Fig. 6. Average accuracy, precision, and recall of the solutions found at every generation of five runs of ECL-LSDf.

generation, and Fig. 7(b) the average and best fitness at each generation. The fine initialization of inequalities with BP intervals allows the algorithm to progressively enlarge the boundaries of inequalities and correctly classify more and more examples until a solution is found.

Thus, ECL-LSDf seems the best choice for handling this type of problems. However, we will see in the next sections that on real-life datasets, ECL-LSDc yields the best accuracy.

## B. Propositional Datasets

In this section, we test the four ECL variants on the ten propositional datasets described in Table I, which are publicly available from the UCI Repository of machine learning databases [18]. In the last column of the table, the number of facts that form the background knowledge of each dataset is given.

Table II contains the parameter settings of the experiments for each dataset, obtained after performing a small number of runs, in the order of ten, on the training sets with ECL-LUD. The choice of ECL-LUD for tuning the parameters is motivated by the fact that we focus on the effect of the discretization methods, so we would like to use a common parameter setting for ECL, which is not biased toward one of the local discretization methods. In all the experiments a maximum of ten applications of generalization/specialization in the optimization procedure are performed.

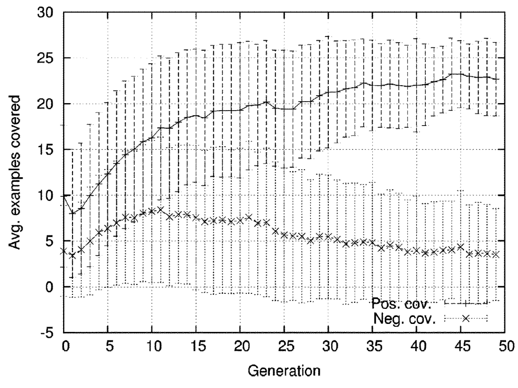
In Table II, $pbk$ controls the probability that each fact in the BK has of being selected and used at each iteration. We have performed several experiments in order to verify if a best value of $pbk$ can be set, but from the results it emerged the domain dependency of $pbk$, whose value has then to be experimentally tuned. We emphasize that the parameter settings chosen were the ones which led to the best classification accuracy in the training set, i.e., the test set was never accessed during the runs allocated for parameter setting.

We use tenfold cross validation, where each dataset is divided in ten disjoint sets of similar size, and the algorithm is run ten times. In each run, one of these ten sets forms the test set, and the union of the remaining nine the training set. Each ECL variant is run three times, using different random seeds, on each training set and its output `Prolog` program is evaluated on the corresponding test set (so each algorithm is run 30 times per dataset).

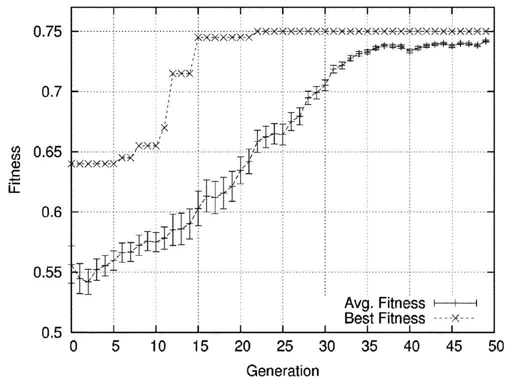
Table IV reports the results of the experiments on the test sets. ECL-LSDc achieves the best accuracy in most of the cases, with simplicity (that is, the number of clauses of the output program) that is second best after ECL-GSD. ECL-LSDf produces best results on the echocardiogram and hepatitis datasets, ECL-GSD on Glass2, but the results are only slightly better than those of ECL-LSDc. The unsupervised variant ECL-LUD produces satisfactory approximate solutions, yet of quality inferior to that of the other methods. The training time of the four algorithms is comparable, where ECL-LSDc and ECL-GSD are slightly faster than the other variants. The average times employed by ECL-LSDc, ECL-LSDf, ECL-LUD, and ECL-GSD on the propositional datasets are, in seconds: 2095.36, 2266.17, 2738.38, and 2112.32, respectively. These times were computed on a Sun Ultra 250, UltraSPARC-II 400 MHz. Table III contains the total number of DP and BP points of the datasets, showing that in general the latter is much bigger than the former.

In order to summarize the performance of the four variants and the significance of the results with respect to the accuracy, we compute the ranking and the statistical paired two-tailed t-test with confidence level of 1% and 5%. The t-test is performed on the 30 results obtained from the ten folds and the three random seeds.

From Table V, we can extract the following hierarchy of the methods: ECL-LSDc, ECL-LSDf, ECL-GSD, and ECL-LUD. Using 1% confidence level, we get that ECL-LSDc is never outperformed, while it is significantly better than the other methods on the Pima-Indians dataset, better than ECL-GSD on the breast dataset, and better than ECL-LUD on the Ionosphere dataset, together with ECL-LSDf and ECL-GSD. If we increase the confidence level to 5%, then we get that ECL-LUD and ECL-LSDc are significantly better than ECL-LSDf on the



(a)



(b)

Fig. 7. Graphs for five runs of ECL-LSDf. Vertical bars show standard deviation. In (b) an accuracy of 0.75 means that an individual correctly identifies a sector of Fig. 1. (a) Average coverage of individuals. (b) Average and best fitness of individuals.

TABLE I
CHARACTERISTICS OF THE DATASETS. FROM LEFT TO RIGHT: NUMBER OF EXAMPLES (POSITIVE, NEGATIVE), OF CONTINUOUS ATTRIBUTES, OF NOMINAL ATTRIBUTES, AND OF ELEMENTS OF THE BK

| Dataset | Examples (+,-) | Continuous | Nominal | BK Dim. |
|---|---|---|---|---|
| Australian | 690 (307,383) | 6 | 8 | 9660 |
| Breast | 699 (458,241) | 10 | 0 | 6275 |
| Crx | 690 (307,383) | 6 | 9 | 10283 |
| Echocardiogram | 74 (24,50) | 5 | 1 | 750 |
| German | 1000 (700,300) | 24 | 0 | 24000 |
| Glass2 | 163 (87,76) | 9 | 0 | 1467 |
| Heart | 270 (120,150) | 13 | 0 | 3510 |
| Hepatitis | 155 (123,32) | 6 | 13 | 2778 |
| Ionosphere | 351 (225,126) | 34 | 0 | 11934 |
| Pima-Indians | 768 (500,268) | 8 | 0 | 6144 |

TABLE II
PARAMETER SETTINGS USED IN THE EXPERIMENTS: *GEN* IS THE NUMBER OF GENERATIONS PERFORMED BY THE GA, *SEL* IS THE NUMBER OF INDIVIDUALS SELECTED PER GENERATION, $N_I, I \in [1,4]$, ARE THE GREEDINESS PARAMETERS OF THE MUTATION OPERATORS, *LC* IS THE MAXIMUM LENGTH OF A CLAUSE, AND *PBK* IS THE PROBABILITY OF SELECTING A BK FACT

| Dataset | pop size | gen | sel | max_iter | $N_i$ | lc | pbk |
|---|---|---|---|---|---|---|---|
| Australian | 50 | 10 | 15 | 1 | (4,4,4,4) | 6 | 0.4 |
| Breast | 50 | 5 | 5 | 1 | (3,3,3,3) | 5 | 1.0 |
| Crx | 80 | 20 | 15 | 1 | (4,4,4,4) | 7 | 1 |
| Ecochardiogram | 40 | 8 | 10 | 10 | (4,4,4,4) | 4 | 0.7 |
| German | 200 | 10 | 10 | 2 | (3,4,3,4) | 6 | 0.4 |
| Glass2 | 150 | 15 | 20 | 3 | (2,8,2,9) | 5 | 0.8 |
| Heart | 50 | 10 | 15 | 1 | (4,4,4,4) | 6 | 1 |
| Hepatitis | 50 | 10 | 10 | 5 | (4,4,4,4) | 7 | 0.2 |
| Ionosphere | 50 | 10 | 15 | 6 | (4,8,4,8) | 6 | 0.2 |
| Pima-Indians | 60 | 10 | 7 | 5 | (2,5,3,5) | 4 | 0.2 |

TABLE III
TOTAL NUMBER OF DP AND BP POINTS PER DATASET

| Dataset | DP | BP |
|---|---|---|
| Australian | 13 | 810 |
| Breast | 29 | 84 |
| Crx | 13 | 806 |
| Echocardiogram | 5 | 151 |
| German | 30 | 256 |
| Glass2 | 16 | 479 |
| Heart | 10 | 294 |
| Hepatitis | 10 | 192 |
| Ionosphere | 145 | 1359 |
| Pima-Indians | 17 | 1123 |

Australian dataset, ECL-LSDf becomes also significantly better than ECL-GSD on the breast dataset, and ECL-LSDc (together with ECL-GSD) becomes significantly better than ECL-LSDf and ECL-LUD on the German dataset. The other datasets (echocardiogram, glass 2, heart, and hepatitis) are small, and the results of the experiments are not normally distributed, so the t-test cannot be applied.

In general, simple solutions are obtained using Fayyad and Irani discretization applied either prior to induction (ECL-GSD) or in the initialization of the inequalities (ECL-LSDc). The simplicity column of the results also indicates that the solutions produced by ECL-LSDf are in general more complex than those generated by the other methods, due to the initialization of the inequalities to rather small intervals. The best tradeoff between simplicity and accuracy is obtained by ECL-LSDc. In the two cases where ECL-LSDf yields accuracy better than ECL-LSDc, the program induced by ECL-LSDf contains many more rules. ECL-GSD obtains best simplicity on all datasets, but on at least three datasets its accuracy is significantly worse than the one of ECL-LSDc.

In summary, the results of the experiments on these propositional datasets seem to indicate that an effective search strategy for discretizing continuous attributes in an evolutionary learner consists of starting from large intervals for initializing inequalities, and then refine them during the evolutionary process using the boundary points for enlarging and shrinking the intervals. The results also indicate that the supervised methods obtain in general better performance than the unsupervised one. For this reason, we will not consider ECL-LUD in the experiments on relational datasets described in the next section.

## C. Relational Datasets

Table VI shows the characteristics of the relational datasets and Table VII shows the parameter settings used in the experiments, obtained after performing few runs on the training sets using ECL-GSD. As for the experiments performed on the propositional datasets, a maximum of ten optimization steps are performed. These datasets are used as benchmark problems for ILP systems (see, e.g., [19]). Table VIII contains the total number of DP and BP points per dataset.

TABLE IV
RESULTS FOR THE VARIOUS METHODS ON THE PROPOSITIONAL DATASETS: AVERAGE ACCURACY ON THE TEST SETS AND NUMBER OF CLAUSES (SIMPLICITY) WITH STANDARD DEVIATION BETWEEN BRACKETS

| Dataset | System | Accuracy | Simplicity |
|---|---|---|---|
| Australian | ECL-LSDc | **0.85 (0.01)** | 6.10 (2.18) |
| | ECL-LSDf | 0.83 (0.01) | 15.50 (3.69) |
| | ECL-LUD | **0.85 (0.01)** | 16.6 (2.72) |
| | ECL-GSD | 0.84 (0.01) | **3.20 (0.79)** |
| Breast | ECL-LSDc | **0.95 (0.02)** | 8.60 (0.41) |
| | ECL-LSDf | 0.94(0.03) | 13.75 (2.05) |
| | ECL-LUD | 0.94 (0.02) | 14.10 (2.08) |
| | ECL-GSD | 0.93 (0.02) | **6.05 (1.34)** |
| Crx | ECL-LSDc | **0.84 (0.01)** | 4.80 (0.05) |
| | ECL-LSDf | 0.82 (0.02) | 11.90 (3.48) |
| | ECL-LUD | 0.83 (0.02) | 9.80 (3.16) |
| | ECL-GSD | 0.83 (0.01) | **3.70 (0.83)** |
| Echocardiogram | ECL-LSDc | 0.74 (0.01) | 2.60 (0.70) |
| | ECL-LSDf | **0.76 (0.02)** | 10.00 (1.15) |
| | ECL-LUD | 0.65 (0.01) | 11.90 (2.02) |
| | ECL-GSD | 0.69 (0.02) | **1.30 (0.48)** |
| German | ECL-LSDc | **0.74 (0.01)** | 11.70 (0.24) |
| | ECL-LSDf | 0.72 (0.02) | 58.40 (2.40) |
| | ECL-LUD | 0.71 (0.01) | 80.30 (6.46) |
| | ECL-GSD | **0.74 (0.01)** | **5.6 (0.57)** |
| Glass2 | ECL-LSDc | 0.85 (0.01) | 4.2 (1.23) |
| | ECL-LSDf | 0.75 (0.03) | 21.8 (2.66) |
| | ECL-LUD | 0.71 (0.03) | 24.40 (3.89) |
| | ECL-GSD | **0.86 (0.02)** | **2.2 (0.42)** |
| Heart | ECL-LSDc | **0.80 (0.03)** | 4.20 (1.32) |
| | ECL-LSDf | 0.73 (0.01) | 9.20 (3.05) |
| | ECL-LUD | 0.77 (0.02) | 10.90 (2.73) |
| | ECL-GSD | 0.77 (0.02) | **2.50 (0.71)** |
| Hepatitis | ECL-LSDc | 0.83 (0.02) | 7.60 (0.95) |
| | ECL-LSDf | **0.84 (0.04)** | 17.70 (2.15) |
| | ECL-LUD | 0.80 (0.03) | 24.50 (4.06) |
| | ECL-GSD | 0.83 (0.03) | **6.40 (1.20)** |
| Ionosphere | ECL-LSDc | **0.89 (0.02)** | 12.50 (1.48) |
| | ECL-LSDf | 0.88 (0.04) | 45.78 (5.37) |
| | ECL-LUD | 0.78 (0.02) | 88.20 (11.25) |
| | ECL-GSD | 0.87 (0.03) | **9.80 (1.34)** |
| Pima-Indians | ECL-LSDc | **0.76 (0.01)** | 8.40 (1.84) |
| | ECL-LSDf | 0.71 (0.01) | 75.60 (5.23) |
| | ECL-LUD | 0.70 (0.01) | 53.80 (6.66) |
| | ECL-GSD | 0.68 (0.02) | **2.20 (0.63)** |

The mutagenesis dataset [20] originates from the problem in organic chemistry of learning the mutagenic activity of nitroaromatic compounds, described as a binary classification problem.

The traffic dataset [21], [22] describes the task of detecting sections of roads where a traffic problem—an *accident* or a *congestion*—has occurred at a specific time.

The biodegradability dataset [23] originates from the task of predicting the half-life time in water for aerobic aqueous biodegradation of a compound. It consists of four classes: *fast* if the biodegradation time of a compound is up to seven days, *moderate* if the biodegradation time is 1–4 weeks, *slow* if the biodegradation time is 1–6 months, and *resistant* in the other cases.

We consider each class of the traffic and the biodegradability datasets as a separate learning task, thus obtaining a total of seven binary classification problems. For the traffic dataset, we report the results only for two classes. In particular, we report results regarding the classes identifying traffic problems. In Section IV-D, we describe how the theories induced for each of the classification problems are combined for handling the multiclass classification problems.

TABLE V
RESULTS OF THE TWO-TAILED PAIRED T-TEST WITH 1% CONFIDENCE LEVEL: EACH ENTRY CONTAINS THE NUMBER OF DATASETS ON WHICH THE ALGORITHM IN THE ROW IS SIGNIFICANTLY BETTER THAN THE ONE IN THE COLUMN. THE RESULTS OF THE TEST USING 5% CONFIDENCE LEVEL ARE REPORTED BETWEEN BRACKETS WHEN THEY DIFFER FROM THOSE USING 1% CONFIDENCE LEVEL. THE TESTS ARE PERFORMED ON THE ACCURACIES OBTAINED ON TEST SET

| Method | ECL-LSDc | ECL-LSDf | ECL-LUD | ECL-GSD | Total |
|---|---|---|---|---|---|
| ECL-LSDc | - | 1 (2) | 2 (3) | 2 | 5 (8) |
| ECL-LSDf | 0 | - | 1 | 0 (1) | 1 (2) |
| ECL-LUD | 0 | 0 (1) | - | 0 | 0 (1) |
| ECL-GSD | 0 | 0 (1) | 1 | - | 1 (2) |
| Total | 0 | 1 (4) | 4 (5) | 2 (3) | |

TABLE VI
CHARACTERISTICS OF THE RELATIONAL DATASETS. FROM LEFT TO RIGHT: DATASET NAME; TOTAL NUMBER OF EXAMPLES AND, BETWEEN BRACKETS, NUMBER OF EXAMPLES PER CLASS; NUMBER OF CONTINUOUS AND NOMINAL ATTRIBUTES; AND NUMBER OF FACTS IN THE BK

| Dataset | Examples | Continuous | Nominal | BK |
|---|---|---|---|---|
| Mutagenesis | 188 (125,63) | 6 | 4 | 13125 |
| Traffic | 256 (62,66,128) | 3 | 2 | 15770 |
| Biodegradability | 328 (65,120,101,42) | 2 | 4 | 17260 |

TABLE VII
PARAMETERS USED IN THE EXPERIMENTS ON ILP DATASETS

| Dataset | pop size | gen | sel | max_iter | $N_i$ | lc | pbk |
|---|---|---|---|---|---|---|---|
| Mutagenesis | 50 | 10 | 15 | 2 | (4,8,2,8) | 3 | 0.8 |
| Traffic | 30 | 10 | 10 | 1 | (10,2,2,2) | 8 | 1.0 |
| Biodegradability | 50 | 10 | 10 | 1 | (4,4,4,4) | 4 | 1.0 |

TABLE VIII
TOTAL NUMBER OF DP AND BP POINTS PER DATASET

| Dataset | DP | BP |
|---|---|---|
| Mutagenesis | 9 | 116 |
| Accidents | 7 | 121 |
| Congestions | 7 | 118 |
| Bio-Fast | 4 | 190 |
| Bio-Slow | 2 | 257 |
| Bio-Moderate | 2 | 311 |
| Bio-Resistant | 4 | 100 |

In the experiments, we use tenfold cross-validation on all datasets and each ECL variant is run three times with different random seeds except on the biodegradability one, where the same splitting of data as in [23] is applied, consisting of five different tenfold cross-validation sets. Table IX shows the results obtained on the test sets.

On the training set ECL-LSDc yields the best average accuracy on the first three datasets, ECL-LSDf outperforms the other algorithms on the last four datasets, and ECL-GSD yields reasonable results, slightly inferior to those of ECL-LSDc. The training time of the algorithms is comparable. However, the higher complexity of the solutions found by ECL-LSDf, containing on average twice the number of clauses of the other algorithms, penalizes its performance on the test sets.

On the mutagenesis dataset the accuracies obtained on the test sets by the three systems are comparable, with ECL-LSDf performing slightly better than ECL-LSDc and ECL-GSD.

On the traffic dataset the best results are produced by ECL-LSDc, while ECL-LSDf obtains the worst performance.

$$\begin{aligned}
\mathtt{accident(X,T)} :- \;& \mathtt{ocupacion(T,Y,O_Y), ocupacion(T,X,O_X), saturacion(T,X,S_X), secciones\_posteriores(X,Y),}\\
& (-\infty < \mathtt{O_Y} \leq 777.625), (-\infty < \mathtt{S_X} \leq 47.0625), (651.25 < \mathtt{O_X} \leq \infty).\\
\mathtt{accident(X,T)} :- \;& \mathtt{velocidad(T,Y,V_Y), tipo(X,X6), saturacion(T,X,S_X), secciones\_posteriores(Y,X),}\\
& \mathtt{ocupacion(T,X,O_X), tipo(Y,X6)}, (-\infty < \mathtt{V_Y} \leq 53.5), (104.875 < \mathtt{O_X} \leq 839.5), (-\infty < \mathtt{S_X} \leq 46.1875).
\end{aligned}$$

$$\begin{aligned}
\mathtt{congestion(X,T)} :- \;& \mathtt{saturacion(T,X,S_X), ocupacion(T,X,O_X)}, (56.5 < \mathtt{S_X} \leq 77.25), (527.25 < \mathtt{O_X} \leq 740.125).\\
\mathtt{congestion(X,T)} :- \;& \mathtt{ocupacion(T,X,O_X), ocupacion(T,Y,O_Y), secciones\_posteriores(Y,X), tipo(X,rampa\_abandono),}\\
& (64.125 < \mathtt{O_X} \leq 574.875), (779.25 < \mathtt{O_Y} \leq \infty).
\end{aligned}$$

Fig. 8. Some clauses generated by ECL-LSDc on the traffic dataset.

TABLE IX
RESULTS OF EXPERIMENTS ON THE RELATIONAL DATASETS: AVERAGE ACCURACY ON THE TEST SETS AND SIMPLICITY OF THE SOLUTION (STANDARD DEVIATIONS BETWEEN BRACKETS)

| Dataset | System | Accuracy | Simplicity |
|---|---|---|---|
| Mutagenesis | ECL-LSDc | 0.88 (0.01) | 4.61 (0.84) |
| | ECL-LSDf | **0.90 (0.01)** | 7.92 (1.51) |
| | ECL-GSD | 0.89 (0.01) | **2.71 (0.38)** |
| Accidents | ECL-LSDc | **0.95 (0.02)** | **3.55 (0.49)** |
| | ECL-LSDf | 0.87 (0.02) | 15.55 (1.06) |
| | ECL-GSD | 0.91 (0.01) | 4.90 (0.85) |
| Congestions | ECL-LSDc | **0.94 (0.02)** | 3.95 (0.35) |
| | ECL-LSDf | 0.84 (0.01) | 7.20 (0.57) |
| | ECL-GSD | 0.93 (0.01) | **2.65 (0.21)** |
| Bio-Fast | ECL-LSDc | **0.82 (0.01)** | **10.28 (1.83)** |
| | ECL-LSDf | 0.77 (0.01) | 23.72 (2.19) |
| | ECL-GSD | **0.82 (0.03)** | 10.66 (2.30) |
| Bio-Slow | ECL-LSDc | 0.68 (0.02) | **13.50 (2.57)** |
| | ECL-LSDf | **0.70 (0.02)** | 25.40 (2.61) |
| | ECL-GSD | 0.66 (0.01) | 13.80 (2.50) |
| Bio-Moderate | ECL-LSDc | **0.66 (0.01)** | **13.98 (3.57)** |
| | ECL-LSDf | 0.62 (0.04) | 25.02 (3.41) |
| | ECL-GSD | 0.62 (0.05) | 14.64 (2.13) |
| Bio-Resistant | ECL-LSDc | **0.91 (0.01)** | **5.28 (1.21)** |
| | ECL-LSDf | 0.90 (0.02) | 12.56 (3.13) |
| | ECL-GSD | 0.89 (0.01) | 5.73 (2.87) |

TABLE X
RESULTS OF THE TWO-TAILED PAIRED T-TEST WITH 1% CONFIDENCE LEVEL: EACH ENTRY CONTAINS THE NUMBER OF DATASETS ON WHICH THE ALGORITHM IN THE ROW IS SIGNIFICANTLY BETTER THAN THE ONE IN THE COLUMN. THE RESULTS OF THE TEST USING 5% CONFIDENCE LEVEL ARE REPORTED BETWEEN BRACKETS WHEN THEY DIFFER FROM THOSE USING 1% CONFIDENCE LEVEL

| Method | ECL-LSDc | ECL-LSDf | ECL-GSD | Total |
|---|---|---|---|---|
| ECL-LSDc | - | 3(4) | 2(3) | 5(7) |
| ECL-LSDf | 0 | - | 1 | 1 |
| ECL-GSD | 0 | 2 | - | 2 |
| Total | 0 | 5(6) | 3(4) | |

In [21] and [22], a discretization provided by experts in the field was used for the three numerical arguments of the traffic dataset. Using the same discretization, ECL-GSD obtained results that are slightly superior to those obtained using Fayyad and Irani algorithm [on the accidents dataset, the average accuracy on the test and training sets is 0.92 (0.03) and 0.94 (0.02) and the average simplicity is 5.10 (0.93). On the congestions dataset, the average accuracy on the test and training sets is 0.93 (0.02) and 0.95 (0.00) and the average simplicity is 3.23 (0.21)]. The two discretizations produce similar partitions for two of the three attributes.

Fig. 8 shows four clauses generated by ECL-LSDc.

The first clause states that there is an accident on section $X$ at time $T$ if the occupancy rate of $X$ at time $T$ is greater than 651.25, the flow rate of $X$ at time $T$ is less than 47.0625, and on a following section $Y$ at time $T$ the occupancy rate is less than 777.625. This clause illustrates the ability of ECL to handle interdependencies between arguments.

On the biodegradability dataset ECL-LSDc obtains the best results on two of the four binary classification problems, and has slightly inferior performance on the bio-slow class.

Similar to the propositional datasets, we analyze further the accuracy results by means of the statistical paired two-tailed t-test with 1% and 5% confidence levels. The results of the tests are reported in Table X. Also in this case, ECL-LSDc turns out to be the best algorithm. ECL-LSDc is never outperformed, and using 1% confidence level it is significantly better than ECL-LSDf on three datasets (accidents, congestion, and bio-fast), and significantly better than ECL-GSD on two datasets (bio-moderate, bio-resistant). Moreover, ECL-GSD is significantly better than ECL-LSDf on two datasets (bio-fast, congestions), while it is outperformed by ECL-LSDf on the bio-slow dataset.

In summary, for the relational datasets, we can draw the same conclusions as for the propositional ones. Namely, that a good performance in terms of accuracy and simplicity is obtained by embedding in ECL a discretization method which initializes inequalities using Fayyad and Irani algorithm, and then refines the inequalities during the learning process using smaller intervals in order to take into account interdependencies between attributes.

## D. Comparison With Other Systems

Although the focus of this paper is not to globally assess the performance of ECL with discretization, it is nevertheless interesting to briefly compare the results of the various settings of ECL with those obtained by other popular propositional and ILP learners.

In Table XI, we compare the results obtained by ECL on the propositional datasets against the results onbtained by four nonevolutionary systems for ICL, C4.5 [24], Naive Bayes [25], SMO [26], and IB1 [27], and two evolutionary algorithms, HIDER* [28] and GAssist [29]. C4.5 is a decision tree algorithm, Naive Bayes uses the Baye's rule of conditional probabilities to estimate the predicted class, SMO implements the sequential minimal optimization algorithm for training a support vector classifier, and IB1 uses a simple distance measure to find the training instance closest to the given test instance, and predicts the same class as this training instance. For these systems, we used the Weka [30] implementation

TABLE XI
AVERAGE ACCURACIES ACHIEVED BY VARIOUS PROPOSITIONAL LEARNERS. STANDARD DEVIATION BETWEEN BRACKETS

| Dataset | ECL | C4.5 | NaiveBayes | SMO | HIDER* | GAssist | IB1 |
|---|---|---|---|---|---|---|---|
| Australian | 0.85 (0.02) | 0.85 (0.04) | 0.77 (0.01) | 0.85 (0.01) | 0.85 (0.03) | 0.85 (0.05) | 0.81 (0.01) |
| Breast | 0.96 (0.02) | 0.94 (0.02) | 0.96 (0.01) | 0.96 (0.01) | 0.97 (0.02) | 0.96 (0.02) | 0.95 (0.01) |
| Crx | 0.84 (0.01) | 0.85 (0.04) | 0.76 (0.01) | 0.85 (0.02) | 0.83 (0.05) | 0.86 (0.05) | 0.81 (0.01) |
| Echocardiogram | 0.74 (0.01) | 0.71 (0.01) | 0.75 (0.02) | 0.75 (0.03) | 0.79 (0.13) | 0.72 (0.02) | 0.67 (0.03) |
| German | 0.74 (0.01) | 0.72 (0.04) | 0.75 (0.02) | 0.76 (0.01) | 0.73 (0.04) | 0.72 (0.02) | 0.67 (0.01) |
| Glass2 | 0.85 (0.01) | 0.78 (0.04) | 0.63 (0.01) | 0.65 (0.02) | 0.79 (0.03) | 0.82 (0.08) | 0.78 (0.01) |
| Heart | 0.80 (0.03) | 0.77 (0.04) | 0.83 (0.01) | 0.83 (0.01) | 0.78 (0.08) | 0.80 (0.07) | 0.76 (0.02) |
| Hepatitis | 0.83 (0.02) | 0.79 (0.04) | 0.83 (0.01) | 0.85 (0.02) | 0.83 (0.02) | 0.89 (0.08) | 0.81 (0.02) |
| Ionosphere | 0.89 (0.02) | 0.89 (0.07) | 0.82 (0.01) | 0.88 (0.02) | 0.89 (0.06) | 0.93 (0.04) | 0.87 (0.01) |
| Pima-Indians | 0.77 (0.01) | 0.73 (0.03) | 0.75 (0.01) | 0.77 (0.01) | 0.74 (0.02) | 0.74 (0.02) | 0.71 (0.01) |

with default parameter settings. Both HIDER* and GAssist are briefly described in Section V.

From the results, it emerges that ECL performs generally better than C4.5 and IB1 on the propositional datasets. Naive Bayes obtains better results on three datasets and in two of these cases the results are comparable. Naive Bayes is outperformed by ECL in five cases, and in some cases, e.g., on Glass2, the difference of accuracy is evident. On the same dataset, ECL outperforms SMO as well, while on the other cases the results achieved by the two systems are comparable. The results obtained by the evolutionary algorithms are also comparable to the ones achieved by ECL, with GAssist performing slightly better.

For the relational datasets, we have compared the results obtained by ECL with those obtained by Progol [31], [32], Tilde [33], and ICL [34]. Here, we consider the traffic and the biodegradability datasets as multiclass classification problems. In order to do so, ECL is run on each class, using the positive examples of the other classes as negative examples. The obtained rules are then combined and a evaluation procedure similar to the one adopted by CN2 [35] is used for assessing the accuracy of the resulting theory: all rules whose conditions apply to a test example are used and the number of training examples of each class covered by the rules are summed up. The class with the largest sum is assigned to the testing example.

The same procedure is used for Progol.

Progol uses inverse entailment and an AQ approach for inducing first order rules.

Tilde and ICL are part of the ACE-ilProlog data mining system [36].

Tilde is an upgrade of C4.5 toward relational data mining. It builds decision trees that allow to predict the value of a certain attribute in a relation from other information in the database. Unlike most ILP systems it uses the learning from interpretations setting, which aims more specifically at classification than the classical ILP settings.

ICL represents an upgrade of CN2 in order to learn first-order rules. CN2 is a propositional learner, that combines the advantages of the rule learner AQ and of the decision tree learner ID3 [37], i.e., it produces understandable rules and can cope with noisy data.

The three ILP systems allow a more expressive background knowledge, which may contain also theories expressing some *a priori* known structures of the background knowledge.

The results obtained by the four systems on the relational datasets are reported in Table XII. ECL outperforms the other systems on the mutagenesis dataset, while on the traffic and biodegradability datasets the results of ECL are comparable to those of other ILP learners. Moreover, the simplicity of the solutions found by ECL and the other ILP learners is similar. The training time of ECL on the relational dataset (average of 785.54 s), is higher than the one of ICL and Tilde, and is comparable to that of Progol.

TABLE XII
AVERAGE ACCURACIES OBTAINED ON THE RELATIONAL DATASETS BY ECL AND OTHER ILP LEARNERS. STANDARD DEVIATION BETWEEN BRACKETS

| Dataset | ECL | Progol | Tilde | ICL |
|---|---|---|---|---|
| Mutagenesis | 0.90 (0.01) | 0.88 (0.02) | 0.86 (0.03) | 0.88 (0.08) |
| Traffic | 0.93 (0.02) | 0.94 (0.03) | 0.94 (0.04) | 0.93 (0.04) |
| Biodegradability | 0.55 (0.03) | 0.53 (0.02) | 0.52 (0.03) | 0.55 (0.02) |

In summary, the experimental comparison suggests that ECL is competitive with state-of-the-art propositional and ILP systems with respect to accuracy and simplicity of the solutions generated by the systems.

## V. RELATED WORK

Recent methods based on evolutionary algorithms performing multivariate discretization during the learning process are [29], [38], and [39], where the evolutionary algorithms for classification GIL [40] and GABIL [41] are extended into the systems EDRL-MD and GAssit, respectively. In both EDRL-MD and GAssist, an individual encodes a set of rules, and continuous attributes are handled by means of inequalities that can be modified during the evolutionary process.

In EDRL-MD, candidate solutions are encoded by means of string chromosomes. The string is composed by $n$ substrings, each encoding a condition related to one attribute. In case of continuous attributes, the relative substring encodes the lower and the upper thresholds of an interval describing the allowed subrange of values for the described attribute. GAssist evolves individuals that are ordered variable-length rule sets. The knowledge representation for real-valued attributes is called adaptive discretization intervals (ADIs) rule representation. This representation uses the same form of rules as GABIL (conjunctive normal form) but uses nonstatic intervals formed by joining several neighbor discretization intervals. The representation can also combine several discretizations at the same time, allowing the system to choose the correct discretizer for each attribute.

Another EA system adopting a similar method for dealing with numerical values is HIDER* [28]. HIDER* is an extension of HIDER [42]–[44], and utilizes the USD [45] discretizer in order to find a number of boundary points that are used as limits of intervals describing subranges of values for numerical attributes. The USD discretizer divides the domains of continuous attributes in a finite number of intervals with maximum goodness, so that the average-goodness of the final set of intervals will be the highest. The main process is divided in two different parts: first, it calculates the initial intervals by means of projections, which will be refined later, depending on the goodnesses obtained after carrying out two possible actions: to join or not adjacent intervals. The main features of the USD are: it is deterministic, does not need any user-parameter, and its complexity is subquadratic. An important feature of HIDER* is its encoding method: each attribute is encoded with only one gene, reducing considerably the length of the individuals and, therefore, the search space size, making the algorithm faster, while maintaining its prediction accuracy. In this coding, inequalities are represented as natural numbers, and can be easily modified during the evolutionary process.

To the best of our knowledge, the only evolutionary algorithm for ILP that adopts some dynamic methods for dealing with numerical values is SIA01 [46]. SIA01 uses intervals for numerical attributes, which are randomly created and modified during the evolutionary process.

Discretization is not the only way to handle real-valued attributes in evolutionary computation applied to ICL. Some examples of alternative ways are induction of decision trees (either axis-parallel or oblique), by either generating a full tree by means of genetic programming operators, as it happens in GALE [47], [48] or using a heuristic method to generate the tree and later a genetic algorithm or an evolutionary strategy to optimize the test performed at each node [49]. Another example is represented by the XCS system [50], [51]. XCS induces rules with real-valued intervals represented as a $(c_i, r_i)$, where $c_i$ and $r_i$ are real numbers, which represents the center and radius of the interval $[c_i - r_i, c_i + r_i]$. Another strategy is generating an instance set used as the core of a *k-NN* classifier [47].

In this paper, we have used discretization algorithms based on Fayyad and Irani's method. In the literature, several other discretization algorithms are reported. Among these the Mántaras discretizer [52] which is similar to the Fayyad and Irani's algorithm, but uses a different formulation of the entropy minimization. Another example of discretization method similar to the Fayyad and Irani's [11], but not relying on entropy, is represented by the Holte's discretization method [53]. This method, used in IB1 [27], attempts to divide the domain of every continuous attributes into pure bins, each containing a strong majority of one particular class with the constraint that each bin must include at least some prespecified number of instances. Yet, another example is ChiMerge [54]. This discretizer creates an initial pool of cut points containing the real values in the domain to discretize, and iteratively merges neighbor intervals that make true a certain criterion based on the $\chi^2$ statistical test.

## VI. Conclusion

This paper analyzed experimentally the effect of different types of discretization techniques on the performance of the evolutionary ILP system ECL. The results of the experiments indicate that a good technique for treating numeric attributes by means of inequalities employs Fayyad and Irani algorithm for initializing the inequalities when they are introduced in a rule, and uses knowledge based mutation operators for refining the inequalities of a rule during the learning process.

The experimental comparison suggests that ECL-LSDc is competitive with state-of-the-art propositional and ILP systems with respect to accuracy and simplicity of the solutions generated by the systems. However, ECL-LSDc is rather slow, especially when compared with propositional learners. The inefficiency of ECL-LSDc is mainly caused by the evaluation of clauses: in order to evaluate a clause, `Prolog` is called on that clause for every example in the training set. Since the communication protocol actually used is rather simple and not optimized, evaluation requires a high computation time. We intend to optimize the code and develop a faster ad-hoc evaluation procedure.

We conclude by describing some issues which can be addressed in future work.

The type of cut points used in our local discretization methods is based on Fayyad and Irani discretization. However, many other choices are possible (cf., e.g., the survey [7]), like for instance, the statistics-based ChiMerge algorithm [54], or the USD discretizer [45].

We have experimented with two types of initialization of inequalities, a fine and a coarse grain one. Other initializations are possible, for instance the one that randomly chooses boundary points. A possible variant of ECL-LUD could be designed, by changing the way in which the operators adopted for modifying inequalities act, in order to exploit information on the class of examples, turning in this way ECL-LUD into a supervised discretization method. This could be done, e.g., by estimating the density distribution of positive and negative examples inside each cluster, and then use this information when modifying inequalities.

Alternative discretization methods based on evolutionary algorithms can evolve a global discretization of the continuous attributes instead of treating directly continuous attributes by means of inequalities, like we did in this paper. For instance, an evolutionary system could co-evolve two populations, one containing individuals which describe (global multivariate) discretizations of continuous attributes, and another one containing individuals which describe rules. The two populations could interact by means of their fitness function, where the fitness of a discretization would be computed by evaluating the quality of the rules in the actual population when continuous attributes undergo that discretization, and the fitness of a rule would involve the discretizations in the actual population.

The system ECL with the discretization methods described in this paper is available for academic use on request, by sending an e-mail to the authors.

## Acknowledgment

The authors would like to thank M. Keijzer for his contribution to an earlier version of this work, R. Giráldez and J. Aguilar–Ruiz for helping in the experiments with HIDER*, and the referees for their constructive suggestions.

## References

[1] T. Mitchell, *Machine Learning*, ser. Computer Science. New York: McGraw-Hill, 1997.

[2] S. Muggleton and L. D. Raedt, "Inductive logic programming: Theory and methods," *J. Logic Prog.*, vol. 19–20, pp. 669–679, 1994.

[3] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Proc. In. Conf. Mach. Learn.*, 1995, pp. 194–202.

[4] A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, ser. Natural Computing. Berlin, Germany: Springer-Verlag, 2002.

[5] A. Freitas and S. Lavington, *Mining Very Large Databases with Parallel Processing*. Norwell, MA: Kluwer, 1998.

[6] R. Kohavi and M. Sahami, "Error-based and entropy-based discretization of continuous features," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 114–119.

[7] H. Liu, F. Hussain, C. Tan, and M. Dash, "Discretization: An enabling technique," *J. Data Mining Knowl. Discovery*, vol. 6, no. 4, pp. 393–23, 2002.

[8] U. Fayyad and K. Irani, "Multi-interval discretization of continues attributes as preprocessing for classification learning," in *Proc. 13th Int. Joint Conf. Artif. Intell.*. San Mateo, CA, 1993, pp. 1022–1027.

[9] F. Divina and E. Marchiori, "Evolutionary concept learning," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 9–13, 2002, pp. 343–350.

[10] F. Divina, M. Keijzer, and E. Marchiori, "A method for handling numerical attributes in GA-based inductive concept learners," in *Proc. Genetic Evol. Comput. Conf.*, Chicago, IL, Jul. 12–16, 2003, pp. 898–908.

[11] U. Fayyad and K. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Mach. Learn.*, vol. 8, pp. 87–102, 1992.

[12] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. River Edge, NJ: World Scientific, 1989.

[13] F. Divina, M. Keijzer, and E. Marchiori, "Non-universal suffrage selection operators favor population diversity in genetic algorithms," in *Proc. Genetic Evol. Comput. Conf.*, Chicago, IL, Jul. 12–16, 2003, pp. 1571–1573.

[14] F. Divina and E. Marchiori, *Knowledge-Based Evolutionary Search for Inductive Concept Learning*. Berlin, Germany: Springer-Verlag, 2004, ch. III, pp. 237–254.

[15] A. Giordana and F. Neri, "Search-intensive concept induction," *Evol. Comput.*, vol. 3, no. 4, pp. 375–16, 1996.

[16] F. Divina, M. Keijzer, and E. Marchiori, "Evolutionary concept learning with constraints for numerical attributes," in *Proc. Belgian–Dutch Conf. Artif. Intell.*, Utrecht, The Netherlands, Oct. 23–24, 2003, pp. 107–114.

[17] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Stat. Soci.y*, vol. 39, pp. 1–38, 1977.

[18] C. Blake and C. Merz. (1998) UCI repository of machine learning databases. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[19] W. Van Laer, "From prepositional to first order logic in machine learning and data mining—induction of first order rules with ICL," Ph.D. Dissertation, Dept. Comput. Sci., K. U. Leuven, Leuven, Belgium, Jun. 2002.

[20] A. Debnath, R. L. de Compadre, G. Debnath, A. Schusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity," *J. Med. Chem.*, vol. 34, no. 2, pp. 786–797, 1991.

[21] S. Dzeroski, N. Jacobs, M. Molina, and C. Moure, "ILP experiments in detecting traffic problems," in *Proc. Eur. Conf. Mach. Learn.*, 1998, pp. 61–66.

[22] S. Dzeroski, N. Jacobs, M. Molina, C. Moure, S. Muggleton, and W. V. Laer, "Detecting traffic problems with ILP," in *Proc. Int. Workshop Inductive Logic Program.*, 1998, pp. 281–290.

[23] S. Dzeroski, H. Blockeel, B. Kompare, S. Kramer, B. Pfahringer, and W. V. Laer, "Experiments in predicting biodegradability," in *Proc. Int. Workshop Inductive Logic Program.*, 1999, pp. 80–91.

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

[25] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proc. 11th Conf. Uncertainty Artif. Intell.*, 1995, pp. 338–345.

[26] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," *Adv. Kernel Methods: Support Vector Learn.*, pp. 185–208, 1999.

[27] D. W. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, pp. 37–66, 1991.

[28] R. Giráldez, J. S. Aguilar-Ruiz, and J. C. Riquelme, "Natural coding: A more efficient representation for evolutionary learning," in *Proc. Genetic Evol. Comput. Conf.*, Chicago, IL, July 12–16, 2003, pp. 979–990.

[29] J. Bacardit and J. M. Garrel, "Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based genetic learning classifier system," in *Proc. Genetic Evol. Comput. Conf.*, 2003, pp. 1818–1831.

[30] I. Witten and E. Frank, *WEKA Machine Learning Algorithms in Java*. San Mateo, CA: Morgan Kaufmann, 2000, ch. 8, pp. 265–320.

[31] S. Muggleton, "Inverse entailment and Progol," *New Generation Comput. (Special Issue on Inductive Logic Programming)*, vol. 13, no. 3–4, pp. 245–286, 1995.

[32] ——, "Learning from positive data," in *Proc. 6th Int. Workshop Inductive Logic Program.*, S. Muggleton, Ed., 1996, pp. 225–244.

[33] H. Blockeel and L. D. Raedt, "Top-down induction of first-order logical decision trees," *Artif. Intell.*, vol. 101, no. 1–2, pp. 285–297, 1998.

[34] L. De Raedt and W. Van Laer, "Inductive constraint logic," in *Lecture Notes in Artificial Intelligence*. Berlin, Germany: Springer-Verlag, 1995, vol. 997, Proc. 6th Conf. Algorithmic Learning Theory, pp. 80–94.

[35] P. Clark and R. Boswell, "Rule induction with CN2: Some recent improvements," in *Proc. 5th Eur. Working Session Learn.*, 1991, pp. 151–163.

[36] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Mon, and H. Vandecasteele, "Improving the efficiency of inductive logic programming through the use of query packs," *J. Artif. Intell. Res.*, vol. 16, pp. 135–166, 2002.

[37] J. R. Quinlan, "Induction of decision trees," *Readings in Mach. Learn.*, vol. 1, pp. 81–106, 1986. Originally published in Machine Learning.

[38] W. Kwedlo and M. Kretowski, "An evolutionary algorithm using multivariate discretization for decision rule induction," *Principles Data Mining Knowl. Discovery*, pp. 392–397, 1999.

[39] J. Bacardit and J. M. Garrell, "Evolution of multi-adaptive discretization intervals for a rule-based genetic learning system," in *Lecture Notes in Artificial Intelligence*. Berlin, Germany: Springer-Verlag, 2002, vol. 2527, Proc. Iberoamerican Conf. Artif. Intell. (IBERAMIA'2002), pp. 350–360.

[40] C. Janikow, "A knowledge intensive genetic algorithm for supervised learning," *Mach. Learn.*, vol. 13, pp. 198–228, 1993.

[41] K. D. Jong, W. Spears, and D. Gordon, "Using genetic algorithms for concept learning," *Mach. Learn.*, vol. 13, no. 1/2, pp. 155–188, 1993.

[42] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "Evolutionary learning of hierarchical decision rules," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 33, no. 2, pp. 324–331, Apr. 2003.

[43] J. C. Riquelme, J. S. Aguilar-Ruiz, and C. D. Valle, "Supervised learning by means of accuracy-aware evolutionary algorithms," *Inf. Sci.*, vol. 156, no. 3–4, pp. 173–188, 2003.

[44] J. S. Aguilar-Ruiz, "Removing examples and discovering hierarchical decision rules with evolutionary algorithms," *Artif. Intell. Commun.*, vol. 14, no. 4, pp. 231–233, 2002.

[45] R. Giráldez, J. S. Aguilar-Ruiz, J. C. Riquelme, F. Ferrer-Troyano, and D. Rodriguez, "Discretization oriented to decision rules generation," *Frontiers Artif. Intell. Applicat.*, vol. 82, pp. 275–279, 2002.

[46] S. Augier, G. Venturini, and Y. Kodratoff, "Learning first order logic rules with a genetic algorithm," in *Proc. 1st Int. Conf. Knowl. Discovery Data Mining*, U. M. Fayyad and R. Uthurusamy, Eds., 20–21, 1995, pp. 21–26.

[47] X. Llorá and J. M. Garrell *et al.*, "Knowledge-independent data mining with fine-grained parallel evolutionary algorithms," in *Proc. Genetic Evol. Comput. Conf.*, L. Spector *et al.*, Eds., Jul. 7–11, 2001, pp. 461–468.

[48] ——, "Inducing partially-defined instances with evolutionary algorithms," in *Proc. 18th Int. Conf. Mach. Learn.*, 2001, pp. 337–344.

[49] E. Cantu-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 1, pp. 54–68, Feb. 2003.

[50] C. Stone and L. Bull, "For real! XCS with continuous—Valued inputs," *Evol. Comput. J.l*, vol. 11, no. 3, pp. 298–336, 2003.

[51] S. W. Wilson *et al.*, "Generalization in the XCS classifier system," in *Proc. 3rd Annu. Conf. Genetic Program.*, J. R. Koza *et al.*, Eds.. Madison, WI, 22–25, 1998, pp. 665–674.

[52] R. L. De Mántaras, "A distance-based attribute selection measure for decision tree induction," *Mach. Learn.*, vol. 6, no. 1, pp. 81–92, 1991.

[53] R. C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Mach. Learn.*, vol. 11, no. 1, pp. 63–90, 1993.

[54] R. Kerber, "ChiMerge: Discretization of numeric attributes," in *Proc. 10th Nat. Conf. Artif. Intell.*, 1992, pp. 123–127.

**Federico Divina** received the degree in computer science from the Ca' Foscari University of Venice, Venice, Italy, and the Ph.D. degree from the Department of Computer Science, Free University of Amsterdam, The Netherlands, with a dissertation on the use of hybrid evolutionary computation for inductive logic programming.

He was a Visiting Researcher at the Machine Learning Group, University of Seville, Seville, Spain. Since September 2004, he has been a Postdoctoral at Tilburg University, Tilburg, The Netherlands, in the European Project NEW TIES. His research interests include evolutionary computation, machine learning, bioinformatics, and artificial societies.

**Elena Marchiori** received the M.S. and Ph.D. degrees from the Department of Mathematics, University of Padova, Padova, Italy.

She has been working at the Center for Wiskunde en Informatica, University of Leiden, Amsterdam, The Netherlands, and at the University Ca' Foscari of Venice, Venice, Italy. She is currently an Assistant Professor in the Department of Computer Science, Free University Amsterdam, The Netherlands. Her main research interests are in optimization and machine learning, using heuristic algorithms based on computational intelligence. In particular, she is involved in research on pattern classification and feature selection applied to medical data.