Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach Learning Classifier System

Jaume Bacardit and Josep Maria Garrell

Intelligent Systems Research Group
Enginyeria i Arquitectura La Salle,
Universitat Ramon Llull,
Psg. Bonanova 8, 08022-Barcelona,
Catalonia, Spain, Europe. {jbacardit,josepmg}@salleURL.edu

Abstract. Bloat control and generalization pressure are very important issues in the design of Pittsburgh Approach Learning Classifier Systems (LCS), in order to achieve simple and accurate solutions in a reasonable time. In this paper we propose a method to achieve these objectives based on the *Minimum Description Length* (*MDL*) principle. This principle is a metric which combines in a smart way the accuracy and the complexity of a theory (rule set , instance set, etc.). An extensive comparison with our previous generalization pressure method across several domains and using two knowledge representations has been done. The test show that the *MDL* based size control method is a good and robust choice.
Keywords : bloat control, generalization pressure, Pittsburgh approach LCS

1 Introduction

The application of Genetic Algorithms (GA) [1, 2] to classification problems is usually known as Genetic Based Machine Learning (GBML), and traditionally it has been addressed from two different points of view: the Pittsburgh approach (or Pittsburgh LCS) and the Michigan approach (or Michigan LCS), early exemplified by LS-1 [3] and CS-1 [4], respectively. Some representative systems of each approach are *GABIL* [5] and XCS [6].

The Pittsburgh approach systems usually evolve variable-length individuals which are complete solutions to the classification problem. This paper deals with the control of the individuals length. This control is a very important issue for two main reasons. The first one is that the evolution of variable-length individuals can lead to solutions growing without control. This phenomenon is usually known as *Bloat* [7] and it has been widely studied in the Genetic Programming field.

The second reason is derived from the fact that usually the fitness of the individuals is only based on their predictive accuracy over the training examples, and doesn't take into account their complexity. Given this fitness function, the easiest way to increase it is to maximize the probability of correctly classifying the train examples, which is achieved by increasing the size of the individuals. This fact produces solutions that are bigger than necessary, contradicting the *Occam's razor* principle [8] which says that "the simplest explanation of the observed phenomena is most likely to be the correct one". A probable consequence of the "over-complexity" is an over-fitting of the solutions created which can lead to a decrease of the generalization capacity. We observed this problem in our previous work [9].

In this paper we propose a bloat control and generalization pressure method (*GPM*) based on the *Minimum Description Length* (*MDL*) principle [10]. It is an interpretation of the Occam's Razor principle based on the idea of data compression, that takes into account both the simplicity and predictive accuracy of a theory. Pfahringer [11] did a very good and brief introduction of the principle:

> Concept membership of each training example is to be communicated from a sender to a receiver. Both know all examples and all attributes used to describe the examples. Now what is being transmitted is a theory (set of rules) describing the concept and, if necessary, explicitly all positive examples not covered by the theory (the false-negative examples) and all negative examples erroneously covered by the theory (the false-positive examples). Now the cost of a transmission is equivalent to the number of bits needed to encode a theory plus its exceptions in a sensible scheme. The MDL principle states that the best theory derivable from the training data will be the one requiring the minimum number of bits.

The *MDL* principle is integrated into our Pittsburgh LCS adapting it to two knowledge representations. The classic *GABIL* one [5] for discrete attributes and our own *Adaptive Discretization Intervals (ADI) rule representation* [12] for the real-valued ones. We have also added an adaptive heuristic in order to simplify the task of domain specific parameter tuning. The *GPM* based on the *MDL* principle is compared across several domains with our previous work in this area: The hierarchical selection operator [9], which is explained in section 3.

The paper is structured as follows. Section 2 presents a short description of how *Bloat effect* affects Pittsburgh *LCS* and also some guidelines about how should the measures to solve the *Bloat* effect be defined. Next, section 3 presents some related work. After the related work we describe the framework of our classifier system in section 4. Our implementation of the *MDL* is explained in section 5. Next, section 6 describes the test suite used in the comparison. The results obtained are summarized in section 7. Finally, section 8 discusses the conclusions and some further work.

2 Bloat effect in Pittsburgh approach LCS

In this section we will do a brief and illustrative introduction about how and why the bloat effect affects Pittsburgh approach LCS. We will also show that fixing this problem is not a simple task, showing how bad ways to fix this problem can collapse the learning process.

2.1 What form does it take the bloat effect?

Usually the bloat effect is defined as the growth without control of the individuals length, and it is a phenomenon that can affect in general all variable-length representations. In Pittsburgh LCS this effect takes the form of an exponential-rate growing of the number of rules of the individuals. This effect can be illustrated by the first 15 iterations in figure 1, which represents the evolution of the average individual size for the *MX11* problem. If we did not apply any measure to control this, the program would crash from out of memory shortly after.

2.2 Why do we have bloat effect?

The reason of the bloat effect is well explained in [7]. Its cause is the use of a fitness function which only takes into account the goodness of the solution (accuracy in our case). Having a variable-length representation means that it possible to have several individuals with the same fitness value, and there will be more long representations of a given solution (fitness value) that short ones. So, when the exploration finds new solutions, it is more probable that these solutions will be long than short.

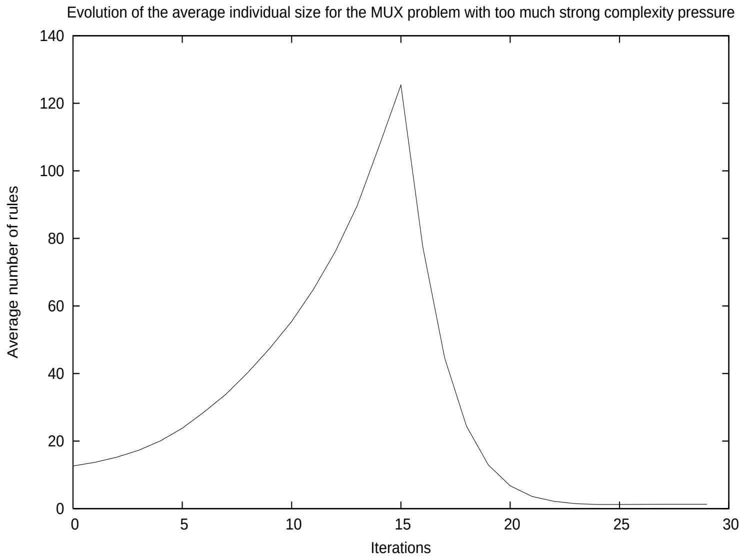
The interpretation of this idea in *LCS* is that, it is more probable to classify correctly more train examples with an individual with a lot of rules that with a short individual. Is this long individual a good solution? Probably no, as this individual is *memorizing* the train examples instead of *learning* them. This shows a side effect of bloat in *LCS*: the generated solutions will probably lack generalization, and its test accuracy will probably be poor.

2.3 How can we solve the bloat effect?

It is obvious that we need to add to the system some bias towards good but simple solutions, but will any intervention in this sense work? The answer is no. If we introduce too much pressure towards finding simple solutions, we are in danger of collapsing the population into individuals of only one rule, which can not generate longer individuals anymore. With this kind of individuals we can only classify the majority class. Again in figure 1 we can see an example of a too much strong pressure in the MX11, which is activated just after 15 iterations. With only a few iterations, a population of an average of more than 120 rules per individual is reduced to one rule individuals. The bloat control method that created this situation is the same presented in this paper, but bad parametrized (InitialRateOfComplexity=0.5).

So, what is the good way to control the bloat effect? There is not a single answer and, beside the method presented in this paper, in the related work section several methods that achieve this control are described. Intuitively we can say that the best method will be the one finding the best balance between accuracy and complexity.

Fig. 1. Illustration of the bloat effect and how a badly designed bloat control method can destroy the population



3 Related work

The *MDL* principle has been applied as a part of modeling tasks in many different fields. For example, handwriting recognition and robotic arms [13]. The principle has also been widely applied in the Machine Learning field. Some examples are the creation of decision trees by means of Inductive Learning [14] , Genetic Programming [15] or, probably, the best known case: c4.5rules [16] , where the *MDL* principle is used to select the best subset of rules derived from a c4.5 induced decision tree.

There is extensive prior work in the Evolutionary Computation (EC) field to control the *bloat* effect, specially in Genetic Programming [15, 17] where this effect has been more widely studied. However, it has also been studied in other EC paradigms like Genetic Algorithms [18, 9, 19] or Evolution Strategies [19]. There is also some work on generalization pressure operators in systems that not suffer the *bloat* effect [20].

4 Framework

In this section we describe the main features of our classifier system. GAssist (*Genetic Algorithms based claSSIfier sySTem*) [21] is a Pittsburgh style classifier system based on GABIL [5]. Directly from GABIL we have borrowed the semantically correct crossover operator.

4.1 General operators and policies

Matching strategy The matching process follows a "if ... then ... else if ... then..." structure, usually called *Decision Lists* [22].

Mutation operators The system manipulates variable-length individuals, making more difficult the tuning of the classic gene-based mutation probability. In order to simplify this tuning, we define p_{mut} as the probability of mutating an individual. When an individual is selected for mutation (based on p_{mut}), a random gene is chosen inside its chromosome to be mutated.

Policy for missing values Some of the problems used in the experimentation reproduced in this paper have missing values. A substitution policy has been used. Before starting the learning process all missing values are changed with either the average value of the attribute (for real-valued attributes) or the most frequent value (for symbolic attributes). These averages are not computed using all the train instances, but only the ones belonging to the same class as the instance with a missing values being substituted.

Pseudo-Random Numbers Generator: The pseudo-random number generator (PRNG) used is the *Mersene Twister* [23], which is one of the best PRNGs currently available. Its application to *GAs* has been studied recently [24] showing that it reduces the fluctuation of the obtained results. The seeds used to initialize the PRNG are taken from pure random data downloaded from www.random.org.

4.2 Bloat control an generality pressure:

We describe briefly our previous work in this area because the *MDL* method presented in this paper will be compared to it in the results section. The bloat control an generality pressure was achieved by combining the following two techniques:

- *Rule deletion:* This operator deletes the rules of the individuals that do not match any training example. This rule deletion is done after the fitness computation and has two constraints: (a) the process is only activated after a predefined number of iterations, to prevent a massive diversity loss and (b) the operator stops when the number of rules of the individual reaches a certain lower threshold.
- *Selection bias using the individual size:* Selection is guided as usual by the accuracy of the individual. However, it also gives certain degree of relevance to the size of the individuals, having a policy similar to multi-objective systems. We use tournament selection because its local behavior lets us implement this policy. The criterion of the tournament is given by our own operator called "hierarchical selection" [9], defined as follows:
  - If $|accuracy_a - accuracy_b| < threshold$ then:
    - If $length_a < length_b$ then a is better than b

  * If $length_a > length_b$ then b is better than a
  * If $length_a = length_b$ then we will use the general case
- Otherwise, we use the general case: we select the individual with higher fitness.

4.3 Knowledge Representations

The following paragraphs describe the knowledge representations that we use to solve problems with symbolic or real-valued attributes. Some of these representations are well known or have been described in detail elsewhere, but we believe that it is important to describe them again because the *MDL* principle has to be carefully adapted for each of them.

Rule Representations for symbolic or discrete attributes We will use the **GABIL** [5] representation for this kind of attributes. Each rule consists of a condition part and a classification part: $condition \rightarrow classification$. Each condition is a Conjunctive Normal Form (CNF) predicate defined as:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee \ldots A_n = V_m^b))$$

Where A_i is the ith attribute of the problem and V_i^j is the jth value that can take the ith attribute.

This kind of predicate can be encoded into a binary string in the following way: if we have a problem with two attributes, where each attribute can take three different values {1,2,3}, a rule of the form "If the first attribute has value 1 or 2 and the second one has value 3 then we assign class 1" will be represented by the string 110|001|1.

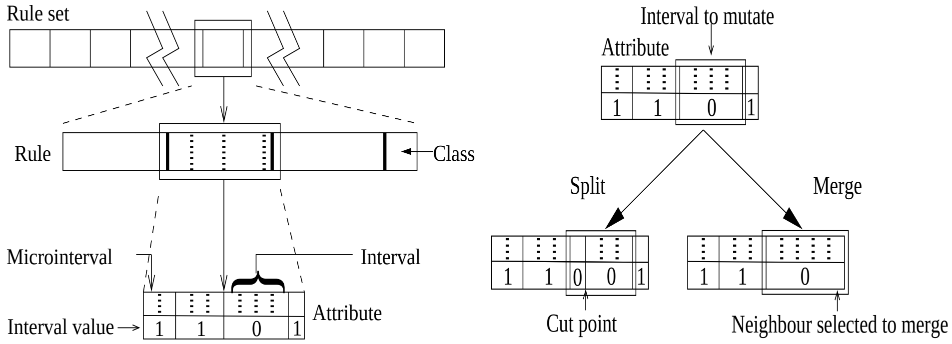
Rule Representations for real-valued attributes The representation for real-valued attributes is our own representation called **Adaptive Discretization Intervals** rule representation [25]. Specifically, we will use the second version of the representation (ADI2) [12].

This representation is an evolution of the *GABIL* discrete rule representation. In *GABIL* for each attribute we would use a set of static discretization intervals instead of nominal values. The intervals of the ADI2 representation are not static, but they evolve through the iterations splitting and merging among them (having a minimum size called *micro-interval*). Thus, the binary coding of the *GABIL* representation is extended as represented in figure 2, also showing the split and merge operations.

The ADI2 representation is defined in depth as follows:

1. Each individuals initial rule and attribute term is assigned a number of "low level" uniform-width and static discretization intervals (called *micro-intervals*).
2. The intervals of the rule are built joining together adjacent *micro-intervals.*

Fig. 2. Adaptive intervals representation and the split and merge operators.



3. Attributes with different numbers of *micro-intervals* can coexist in the population. The evolution will choose the correct number of *micro-intervals* for each attribute.
4. For computational cost reasons, we will have an upper limit in the number of intervals allowed for an attribute, which in most cases will be less than the number of *micro-intervals* assigned to each attribute.
5. When we split an interval, we select a random point in its *micro-intervals* to break it.
6. When we merge two intervals, the state (1 or 0) of the resulting interval is taken from the one which has more *micro-intervals*. If both have the same number of *micro-intervals*, the value is chosen randomly.
7. The number of *micro-intervals* assigned to each attribute term is chosen from a predefined set.
8. The number and size of the initial intervals is selected randomly.
9. The cut points of the crossover operator can only take place in attribute terms boundaries, not between intervals. This restriction takes place in order to maintain the semantical correctness of the rules.
10. The *hierarchical selection* operator uses the length of the individuals (defined as the sum of all the intervals of the individual) instead of the number of rules as the secondary criteria. This change promotes simple individuals with more reduced interval fragmentation.

In order to make the interval splitting and merging part of the evolutionary process, we have to include it in the *GA* genetic operators. We have chosen to add to the GA cycle two special stages applied to the offspring population after the mutation stage. The split and merge operators are controlled by a probability (p_{split} and p_{merge}) defined for each attribute term of each rule. The code for the merge operator probability is represented in figure 3.

Fig. 3. Code of the application of the merge operator.

```
ForEach Individual i of Population
        ForEach Rule j of Population individual i
                ForEach Attribute k of Rule j of Population individual i
                        If random [0..1] number < p_merge
                          Select one random interval of attribute term k
                                  of rule j of individual i
                          Apply a merge operation to this interval
                        EndIf
                EndForEach
        EndForEach
EndForEach
```

5 The *MDL* principle applied to generalization pressure

In this section we describe our proposal of bloat control and generalization pressure based on the *MDL* principle. First, we introduce our implementation of the basic formula of the principle and its adaptation to each of the knowledge representations uses. Finally , we propose a method to adjust automatically the W parameter of the main *MDL* formula that appears in the introduction section, simplifying the domain-specific adjusting of the principle.

5.1 Basic *MDL* formula

As said in the introduction section, the *MDL* principle is a metric used to evaluate the complexity and accuracy of a theory which is inspired by data compression. The class membership of each training example is to be communicated from a sender to a receiver. This is done by transmitting a theory (set of rules in our case) and, if necessary, transmitting the exceptions to this theory. That is, the misclassified and non-classified examples. The cost of the transmission is equivalent to the number of bits needed to encode the theory plus its exceptions in a sensible scheme. The principle states that the best theory is the one requiring the minimum number of bits. Therefore, the fitness function becomes the minimization of the *MDL* formula [16]:

$$MDL = W \cdot \text{theory bits} + \text{exception bits} \tag{1}$$

W is a weight that adjust the relation between theory and exception bits. The length of the theory bits (TL) is defined as follows:

$$TL = \sum_{i=1}^{\text{nr}} TL_i \tag{2}$$

Where nr is the number of rules of the theory. The definition of the rules for all the knowledge representations used share a common structure: *condition* $\rightarrow$

class. The condition is defined as a conjunction of predicates, where each predicate is associated to an attribute of the problem. Therefore, TL_i is defined as follows:

$$TL_i = \sum_{j=1}^{\text{na}} TL_i^j. \tag{3}$$

Where na is the number of attributes of the problem. TL_i^j is the length of the predicate associated to the attribute j of the rule i, and has a specific formula for each knowledge representation used. The reader can see that we have omitted a term in the formula related to the class associated to the rule. As it is a value common for all the possible rules it becomes irrelevant and it has been removed for simplicity reasons.

The exceptions part of the *MDL* principle (EL) represents the act of sending the class for the misclassified or unclassified examples to the receiver. We implement this idea by sending the number of exceptions plus, for each exception, its index in the examples set (supposing that sender and receiver have the examples organized in the same order) and its class:

$$EL = log_2(ne) + (nm + nu) \cdot (log_2(ne) + log_2(nc)) \tag{4}$$

Where ne is the total number of examples, nm is the number of wrongly classified examples, nu is the number of unclassified examples and nc is the number of classes of the problem.

5.2 Adaptation of the *MDL* principle for each knowledge representation

The length of the predicate associated to each attribute (TL_i^j) has to be adapted to the type of the attribute and the knowledge representation. While designing the formula to calculate this length we have to remember that the philosophy of the *MDL* principle is to promote simple but accurate solutions. Therefore, we will prefer formula definitions that promote bias towards simpler solutions although there may exist shorter definitions.

***MDL* formula for real-valued attributes and ADI2 rule representation** The predicate associated to an attribute by this representation is defined as a disjunction of intervals, where each interval is a non-overlapping number of *micro-intervals* and can take a value of either true of false. Therefore, the information to transmit is the number of intervals of the predicate plus, for each interval, its size and value (1 or 0).

$$TL_i^j = log_2(\text{MaxI}) + ni_i^j \cdot (log_2(\text{MaxMI}) + 1) \tag{5}$$

$MaxI$ is the maximum number of intervals allowed in a predicate, ni is the actual number of intervals of the predicate and $MaxMI$ is the maximum allowed number of *micro-intervals* in the predicate.

Given the example of attribute predicate in figure 4, where we have 4 intervals , and supposing that the maximum numbers of intervals and *micro-intervals* are respectively 10 and 25, its MDL size is defined as follows:

$$TL_i^j = log_2(10) + 4 \cdot (log_2(25) + 1)$$

Fig. 4. Example of an ADI2 attribute predicate

1	1	0	1

***MDL* formula for discrete attributes and GABIL representation** The predicate associated to an attribute by this representation is defined as a disjunction of all the possible values that can take the attribute. The simpler way of transmitting this predicate is sending the binary string that the representation uses to represent it. This is the approach used by Quinlan in C4.5rules [16]. However, this definition does not take into account the complexity of the term and does not provide a bias towards generalized solutions.

Therefore, we define a different formula which is very similar to the one proposed for the *ADI2* knowledge representation. In this formula we simulate that we have merged the neighbor values of the predicate which have the same value (true or false):

$$TL_i^j = log_2(nv_j) + 1 + ni_i^j \cdot log_2(nv_j) \tag{6}$$

nv is the number of possible values of the attribute j and ni is the number of "simulated intervals" that exist in the predicate. The only difference between this formula and the *ADI2* one is that we do not have to transmit the value of all the "simulated intervals", but only the first one (one bit).

If we had an attribute predicate such as "1111100001" we can see that we have 10 values and 3 "simulated intervals" and that the MDL size of the predicate would be:

$$TL_i^j = log_2(10) + 1 + 3 \cdot log_2(10)$$

This approach completely makes sense for ordinal attributes, where there exist an order between values, but not for nominal ones. However, we think that this definition is also useful for nominal attributes because we want to promote generalized predicates, where most of the values are true, and this means having few "simulated intervals".

5.3 Looking for a parameter-less *MDL* principle

If we examine all the formulas of the *MDL* principle we only find one parameter: W which adjusts the relation between the length of the theory and the length of the exceptions. Quinlan used a value of 0.5 for this parameter in *C4.5rules* and reported the following in page 53 of [16]:

> Fortunately, the algorithm does not seem to be particularly sensitive to the value of W.

Unfortunately, our environment of application of the *MDL* principle (a *GBML* system) is quite different and the value of the W parameter is quite sensitive. If the value of W is too high, the population will collapse into one rule individuals, as it can be seen in section *bloat*. If W is too low, the individuals probably will be too much specific.

This problem with the adjusting of W leads to a question: Is it possible to find a good method to adjust automatically this parameter? The completely rigorous answer, being aware of the *No Free Lunch Theorem* [26] and the *Selective Superiority Problem* [27] is no.

Nevertheless, at least we can try to find a way to automatically make the system perform "quite well" in a broad range of problems. In order to achieve this objective we have developed a simple approximation which starts the learning process with a very strict weight (but loose enough to avoid a collapse of the population) and relaxes it through the iterations when the *GA* has not found a better solution for a certain number of iterations. This method can be represented by the code in figure 5.

$InitialRateOfComplexity$ defines which percentage of the MDL formula should the term $W \cdot TL$ have. Using this supposition and given one individual from the initial population, we can calculate the value of W. We have used a simple policy to select this individual: the one with more train accuracy ($W = \frac{InitialRateOfComplexity \cdot EL}{(1-InitialRateOfComplexity) \cdot TL'}$).

This arises a question, is this individual good enough? If we recall section 2, it is more probable that this individual will be long than short. Then, maybe we would be initializing W with a too mild value. Therefore, before calculating the initial value of W we do a last step: scaling the theory length of this individual ($TL' = TL \cdot \frac{NR}{NC}$), using as a reference the minimum possible number of rules of an optimal solution: the number of classes of the domain.

We can see that in order to automatically adjust one parameter we have introduced three extra parameters (*InitialRateOfComplexity*, *MaximumBestDelay* and *WeightRelaxationFactor*). The second parameter is easy to setup if we consider the takeover time for the tournament selection [28]. Given a tournament size of 3 and a population size of 300, the takeover time is 6.77 iterations. Considering that we have both crossover and mutation in our *GA*, setting $MaximumBestDelay$ to 10 seems quite safe.

Setting $InitialRateOfComplexity$ is also relatively easy: it the value is too high (giving too much importance to the complexity factor of the MDL formula) the population will collapse. Therefore, we have to find the maximum value of

Fig. 5. Code of the parameter-less learning process with automatically adjusting of W

```
Initialize GA
Ind = Individual with best accuracy from the initial GA population
TL = Theory Length of Ind
EL = Exceptions Length of Ind
NR = Number of rules of Ind
NC = Number of Classes of the domain
TL' = TL · NR/NC
W = (InitialRateOfComplexity · EL) / ((1 − InitialRateOfComplexity) · TL')
Iteration = 0
IterationsSinceBest = 0
While Iteration < NumIterations
        Run one iteration of the GA using W in fitness computation
        If a newbest individual has been found then
                IterationsSinceBest = 0
        Else
                IterationsSinceBest = IterationsSinceBest + 1
        EndIf
        If IterationsSinceBest > MaximumBestDelay then
                W = W · WeightRelaxationFactor
                IterationsSinceBest = 0
        EndIf
        Iteration = Iteration + 1
EndWhile
```

$InitialRateOfComplexity$ that lets the system perform a correct learning process. Doing some short tests with various domains we have seen that values over 0.1 are too much dangerous. In order to adjust more finely this parameter and also set *WeightRelaxationFactor* we have done tests using again the *MX-11* domain testing three values of each parameter: 0.1, 0.075 and 0.05 for $InitialRateOfComplexity$ and 0.9, 0.8 and 0.7 for $WeightRelaxationFactor$.

The results can be seen in table 1, showing three things: test accuracy and the number of rules of the best individual in the final population and also the average iteration where 100% train accuracy was reached. We can see that all the tested configuration manage to reach a perfect accuracy, and also that the number of rules of the solutions are very close to the optimum 9 ordered rules. The only significant differences between the combinations of parameters tested comes when we observe the iterations needed to reach 100% train accuracy. We can see that as more mild are the parameters used, fewer iterations are needed. This arises the question of how extrapolative to other domains is this behaviour. We have to be aware that *MX-11* is a synthetic problem without noise.

In order to check how is the system behaving in real problems, we repeated this test with another well-known problem: *Wisconsin Breast Cancer*. The re-

Table 1. Tests with the MX-11 domain done to find the values of InitialRateOfComplexity (*IROC*) and WeightRelaxationFactor (*WRF*)

WRF	IROC	Test acc.	Num. of Rules	Iterations until perfect accuracy
0.7	0.05	100.0±0.0	9.3±0.6	301.4±56.8
	0.075	100.0±0.0	9.2±0.5	309.0±62.6
	0.1	100.0±0.0	9.2±0.5	333.3±62.2
0.8	0.05	100.0±0.0	9.3±0.5	331.0±71.5
	0.075	100.0±0.0	9.2±0.3	364.4±75.3
	0.1	100.0±0.0	9.2±0.5	374.3±66.9
0.9	0.05	100.0±0.0	9.2±0.5	428.6±99.7
	0.075	100.0±0.0	9.2±0.4	475.5±95.6
	0.1	100.0±0.0	9.1±0.4	518.4±110.2

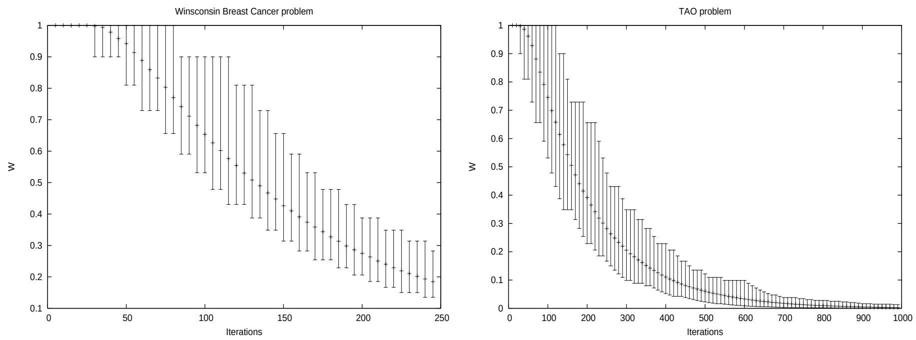
sults can be seen in table 2. Iterations are not included in this table because we do not know the ideal solution for this problem. Instead, we have included train accuracy. It will help illustrate the completely different landscape that we have here: Although the differences are not significant, we can see that as more mild are the parameters used, we have more train accuracy, more rules and less test accuracy. It seems quite clear that the system suffers from over-learning if its working parameters are not enough strict. Therefore, we select 0.075 and 0.9 as the values of $InitialRateOfComplexity$ and $WeightRelaxationFactor$ respectively for the rest of this paper. These values seem to be the most stable ones.

Table 2. Tests with the Wisconsin Breast Cancer domain done to find the values of InitialRateOfComplexity (*IROC*) and WeightRelaxationFactor (*WRF*)

WRF	IROC	Train acc.	Test acc.	Num. of Rules
0.7	0.05	98.2±0.3	95.6±1.5	4.3±1.5
	0.075	98.2±0.3	95.8±1.5	4.1±1.3
	0.1	98.1±0.3	95.9±1.7	3.9±1.2
0.8	0.05	98.1±0.3	95.8±1.5	3.9±1.3
	0.075	98.0±0.3	96.0±1.7	3.7±0.8
	0.1	97.9±0.3	96.0±1.7	3.5±0.9
0.9	0.05	97.8±0.3	95.9±1.7	2.9±0.9
	0.075	97.6±0.3	96.0±1.8	2.3±0.6
	0.1	97.5±0.3	95.9±1.8	2.2±0.5

Before showing the results for all the datasets tested it would be interesting to see the stability of the W tuning heuristic presented in this section. In figure 6 we can see the evolution of W through the learning process for the *bre* and *tao* problems [1]. The values in the figure have been scaled in relation to the initial W value. These two problems are selected because they show two alternative behaviours due to having very different number of rules in their optimal solutions. We can see that the differences in the evolution of W for different executions shrink through the iterations, showing the stability of the heuristic.

Fig. 6. Evolution of W through the learning process



6 Test suite

This section summarizes the tests done across several domains in order to evaluate the accuracy and efficiency of the method presented in this paper. We also compare it with our previous proposal.

6.1 Test problems

The selected test problems for this paper present different characteristics in order to give us a broad overview of the performance of the methods being compared.

First we have some synthetic problems: Tao (*tao*) [29], a problem that has non-orthogonal class boundaries, the 11 input multiplexer (*mux*) and LED (*led*), a problem which represents a seven segments display having the represented digit as the class. This problem has a 10% artificially added noise. Second, we also use several real problems provided by the University of California at Irvine (UCI) repository [30]. The problems selected are: Audiology (*aud*), Glass *gls*, Iris *irs*, Ionosphere (*ion*), Pima-indians-diabetes (*pim*), Primary-Tumor (*prt*) and Wisconsin-breast-cancer (*bre*). Finally, we will use three problems from our own private repository. The first two deal with the diagnosis of breast cancer

[1] Datasets are detailed in section 6

based of biopsies (*bps*) [31] and mammograms (*mmg*) [32] whereas the last one is related to the prediction of student qualifications (*lrn*) [21]. The characteristics of all the datasets are listed in table 3. The partition of the examples into the train and test sets was done using the *stratified ten-fold cross-validation* method [33].

Table 3. Characteristics of the test problems.

Dataset	Number of examples	real-valued attributes	discrete attributes	classes
aud	226	-	69	24
bps	1027	24	-	2
bre	699	-	9	2
gls	214	9	-	6
ion	351	34	-	2
irs	150	4	-	3
led	2000	-	7	10
lrn	648	4	2	5
mmg	216	21	-	2
mux	2048	-	11	2
pim	768	8	-	2
prt	339	-	17	22
tao	1888	2	-	2

6.2 Experimentation design

The goal of the tests done in this paper is to evaluate the performance of the implementation of the *MDL* principle described in the prior section. This evaluation includes a comparison of this method our previous generalization pressure methods (GPM): the *Hierarchical Selection* operator [9].

In our previous work, the *Hierarchical Selection* operator was used in combination with the rule deletion operator because it could not control the bloat effect by itself, but only improved the generalization pressure. This fact makes us question if it is necessary to use the rule deletion operator for the *MDL* methods. We performed a short test to answer this question. The test used again *Wisconsin Breast Cancer*. We use the same GA configuration being used in the global tests which is detailed at the end of this section.

The results of this short test are in table 4. We show, for each configuration (GPM with/without rule elimination), the averages and mean deviations of the test accuracy, the number of rules of the final solution and the run time in seconds (using a Pentium IV at 1.5GHz). We can see that the use of the rule deletion operator improved the accuracy for all the *GPM*. Also, there is a reduction in the average number of rules (for the Hierarchical *GPM*) and run time. The rule set size reduction does not seem very big in average, but the differences are considerable if we look at the maximum and minimum sizes for the Hierarchical method, reflecting that it sometimes cannot control bloat.

Other domains showed similar results. As it seems there does not exist a bad interaction between the GPM and the rule elimination operator, we have decided to use the operator for the rest of the tests.

Table 4. Test of the effects of the Rule Deletion operator for the *Breast* problem

GPM	Rule Deletion	Test accuracy	Number of Rules			Run Time (s)
			Min.	Max.	Avg.	
Hierar.	No	95.6±1.3	22	2	4.9±3.6	92.8±25.6
Hierar.	Yes	95.8±1.6	6	2	2.4±0.7	57.4±2.5
MDL	No	95.9±1.7	6	2	2.4±0.7	58.9±3.4
MDL	Yes	96.1±1.8	4	2	2.4±0.7	55.8±1.9

In order to allow the replication of our results we show the detailed configuration of our tests in table 5. This table is divided in two parts: common and domain-specific parameters.

The value of the initialization probability (p_1) is greater than the usual 0.5 value for some problems. All these problems share a common trait: a high number of attributes. In this environment, a regular initialization policy can lead to a situation where very few (or none) train examples are matched by the individuals. This situation can lead to a collapse of the population towards one rule individuals, because accuracy becomes an insignificant part of the fitness computation.

7 Results

In this section we present the results obtained. The aim of the tests was to determine the performance of the *GPM* tested in three aspects: accuracy and size of the solutions as well as computational cost. For each method and test problem we show the average and standard deviation values of: (1) the cross-validation accuracy, (2) the size of the best individual in number of rules and (3) the execution time in seconds. The tests were executed in an AMD Athlon 1700+ using the Linux operating system, C++ language and GCC v3.2.2 compiler.

The results can be seen in table 6. The results were also analyzed using paired two-sided statistical t-test [34] in order to determine if the *MDL* method outperform our previous approach with a significance level of 1%. No significant outperformances were detected.

As an external reference of the results, in table 7 the accuracy of the two above methods is compared to *IB1* [35], *C5.5* [16] [2] and *XCS* [6] [3]. We can see

[2] Using the Weka [34] implementations
[3] Results taken from [36]

Table 5. Common and problem-specific parameters of the GA.

Parameter	Value
General parameters	
Crossover probability	0.6
Selection Algorithm	Tournament
Tournament size	3
Population size	300
Probability of mutating an individual	0.6
Number of seeds for each experiment	15
MDL Weight heuristically adjusting	
InitialRateOfComplexity	0.075
MaximumBestDelay	10
WeightRelaxationFactor	0.9
rule deletion operator	
Iteration of activation	40
Minimum number of rules before disabling the operator	$numClasses + 3$
Hierarchical Selection	
Iteration of activation	40
ADI rule representation	
Maximum number of intervals per attribute	10
Possible size in *micro-intervals* of an attribute	5,6,7,8,10,15,20,25
p_{split}	0.05
p_{merge}	0.05

Code	Parameter
#iter	Number of GA iterations
p_1	Probability of value 1 in initialization
d_{comp}	Threshold parameter in Hierarchical Selection

Problem	Parameter		
	#iter	p_1	d_{comp}
aud	1500	0.9	0.005
bps	300	0.75	0.015
bre	250	0.5	0.010
gls	1100	0.5	0.010
ion	450	0.75	0.010
irs	200	0.5	0.010
led	1000	0.5	0.001
lrn	700	0.5	0.010
mmg	275	0.75	0.010
mux	1000	0.5	0.001
pim	225	0.5	0.010
prt	1000	0.9	0.005
tao	900	0.5	0.001

that, as usual, each method is the best in some problems but all of them perform similarly in average.

What can we observe in the results? First of all we can see that for the *mux* problem, the *MDL* method manages to generate solutions more near to the optimum rule set than the Hierarchical Selection method. Also, from a global point of view the results tell us that the *MDL* method has achieved our objective of developing a robust and easier to adjust *GPM*. It has managed to outperform (in average) our previous work, the Hierarchical Selection method, in two ways: accuracy and reduction of the computational cost.

Nevertheless, the differences in the results do not seem to be much significant, but the way to reach these results, the internal behaviour of each method, is very

Table 6. Results of the comparative tests. Bold entries show the method with best results for each test problem.

Problem	Configuration	Accuracy	Number of Rules	Time (s)
aud	Hierar.	60.0±4.2	11.2±2.2	89.1±12.4
	MDL	**63.5±3.9**	10.6±2.9	121.2±20.2
bps	Hierar.	**80.2±2.9**	3.4±0.8	218.0±17.3
	MDL	**80.2±2.9**	3.3±1.0	218.9±13.4
bre	Hierar.	95.8±1.6	2.4±0.7	44.6±1.9
	MDL	**96.1±1.8**	2.4±0.7	43.4±1.5
gls	Hierar.	64.4±3.6	7.2±1.4	71.1±7.9
	MDL	**64.8±3.0**	8.7±1.1	74.2±8.2
ion	Hierar.	90.7±2.8	4.0±1.2	177.6±20.8
	MDL	**91.3±2.9**	5.0±1.6	173.7±17.5
irs	Hierar.	95.1±2.1	4.8±1.0	5.3±0.3
	MDL	**95.6±3.0**	4.6±0.8	5.4±0.3
led	Hierar.	74.4±1.7	18.0±2.0	344.3±13.4
	MDL	**74.6±1.7**	19.3±2.2	332.7±8.2
lrn	Hierar.	**68.2±4.6**	7.1±1.6	82.9±5.9
	MDL	68.1±4.1	9.6±2.0	85.4±5.5
mmg	Hierar.	**66.3±4.5**	5.1±1.1	39.9±4.7
	MDL	64.4±6.4	5.3±1.1	38.6±4.7
mux	Hierar.	**100.0±0.0**	10.9±1.1	519.0±36.7
	MDL	**100.0±0.0**	9.2±0.4	474.2±14.4
pim	Hierar.	**75.0±3.4**	4.5±1.3	57.8±5.0
	MDL	74.8±3.4	3.9±0.9	57.4±4.7
prt	Hierar.	46.9±5.3	10.2±2.6	39.4±5.3
	MDL	**47.1±5.2**	14.9±3.5	47.1±5.5
tao	Hierar.	**94.9±1.1**	18.1±3.9	461.3±46.5
	MDL	94.7±0.9	15.1±4.6	414.1±33.0
average	Hierar.	77.8±15.9	8.2±5.0	165.4±164.7
	MDL	78.1±15.8	8.6±5.0	160.5±148.5

Table 7. Accuracy of Hierar. and MDL methods compared to IB1, C4.5 and XCS. Bold entries show the method with best results for each test problem.

Problem	Hierar.	MDL	IB1	C4.5	XCS
aud	60.0±4.2	63.5±3.9	76.0±6.3	**79.0±6.2**	41.6±8.1
bps	80.2±2.9	80.2±2.9	**83.2±3.0**	80.1±4.5	**83.2±2.9**
bre	95.8±1.6	96.1±1.8	96.0±1.4	95.4±1.5	**96.4±2.4**
gls	64.4±3.6	64.8±3.0	66.3±10.4	65.8±9.9	**70.8±8.1**
ion	90.7±2.8	**91.3±2.9**	86.9±4.6	89.8±4.	—
irs	95.1±2.1	**95.6±3.0**	95.3±3.1	95.3±3.1	94.7±5.0
led	74.4±1.7	74.6±1.7	56.5±1.7	**75.0±2.1**	74.5±1.9
lrn	68.2±4.6	68.1±4.1	61.4±5.8	**68.6±4.4**	—
mmg	**66.3±4.5**	64.4±6.4	63.5±11.5	64.8±6.0	64.3±6.4
mux	**100.0±0.0**	**100.0±0.0**	78.6±3.8	99.9±0.2	**100.0±0.0**
pim	75.0±3.4	74.8±3.4	70.3±3.3	73.1±5.0	**75.4±4.4**
prt	46.9±5.3	**47.1±5.2**	37.8±5.3	44.1±5.8	39.9±6.6
tao	94.9±1.1	94.7±0.9	**96.1±1.1**	95.1±1.9	89.9±1.2

different for both methods. We can observe this fact looking at the evolution of the accuracy average individual size (in rules) through the iterations. It figure 7 we can see this evolution for the *bps*,*bre*, *mux* and *tao* problems. The plot for the iterations before the rule deletion activation have been removed from the graph because they introduce a high distortion.

The Hierarchical Selection method uses a specific-to-general policy. In the early iterations of the learning process it frequently finds new solutions that outreach the previous best accuracy by more than d_{comp}. In this situation the number of rules of the individuals is irrelevant. But as the learning curve stabilizes, the differences in accuracy between the bests individuals of the population become smaller than d_{comp}. Then, the smaller individual are mostly selected and, as a consequence, the average individual size slowly decreases.

On the other hand the *MDL* method, because of the behaviour of the W control heuristic, starts the learning process giving much importance to the size of the individual, and relaxes this importance through the iterations as dictated by the heuristic. Therefore, the behaviour is general-to-specific.

In figure 7 we can also see the main problem of the *MDL* method, which is the over-relaxation of the W weight. The philosophy of the algorithm we have proposed to tune W is that we relax this weight when it is too much strict, that is, when the *GA* cannot find a better individual for a certain number of iterations. This condition is sometimes difficult to control, and maybe if the system was given more iterations, the test performance in some domains would decrease. On the other hand we can see in figure 7 and in table 6 the reverse situation for the *tao* problem: The accuracy obtained by the *MDL* method is below the Hierarchical Selection one because the rule set is too much simple. With some more iterations this method probably would increase its accuracy.

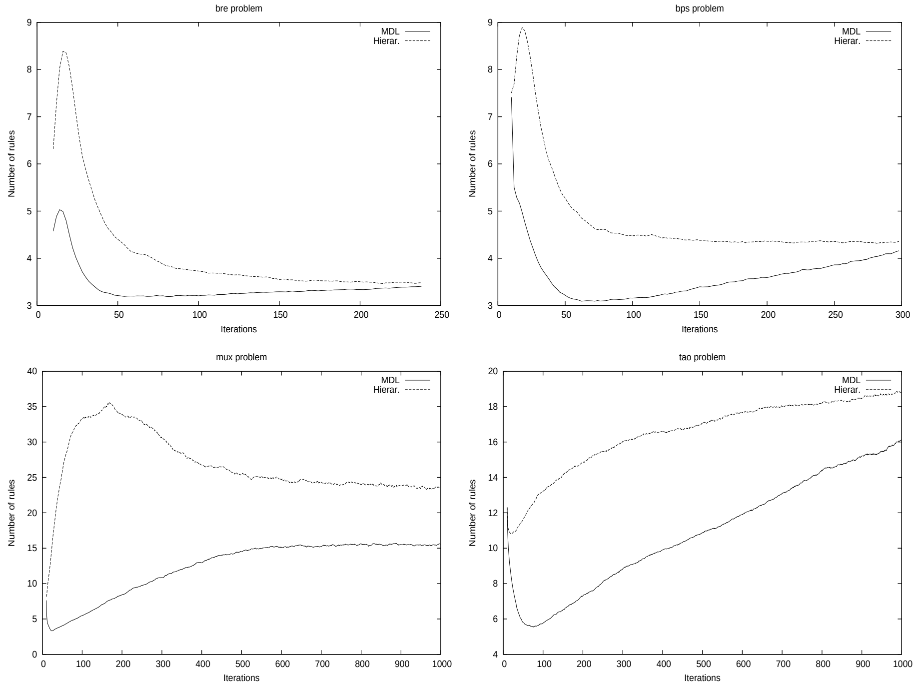
Figure 7 can also help explain the notable computational cost difference between the tested methods in some domains (*mux* and *tao*). Smaller individuals are faster to evaluate. Therefore, the notable differences in the average individual size have their consequence in the overall computational cost.

8 Conclusions and further work

In this paper we have proposed a generalization pressure method for Pittsburgh Approach Classifier Systems based on the *MDL* principle. This technique proposes a fitness function which combines in a smart way the accuracy and the complexity of the individual being evaluated. The complexity measure is not based only on the size of the individual (number of rules) but also on the content of the rules. This if one of the main differences between this method and others found in the literature. Having a bloat control method that takes into account the semantical content of the rules can help explore better the search space, beside managing the size of the individual.

Extensive tests comparing the *MDL* method with our previous proposal have been done. These tests show that the technique performs slightly better (although not in a significant way based on Student t-tests) and runs also slightly faster. Beside its good results, the *MDL* method has another interesting feature, compared to our previous work in *GPM*, which is that it does not need a specific adjustment for each problem being solved. This is due to an adaptive adjustment of the W parameter. This adjustment is done by an heuristic process that we have developed. The adjusting of the W parameter is critical because applying

Fig. 7. Evolution of the average individual size for the *bre* and *bps* problems and *ADI* representation



too much or too little generalization pressure in the population can lead to an incorrect learning process. In the first case the population can collapse into individuals which are too simple. On the other hand, too little pressure can lead to over-fitted solutions. The tests have show that the adjustment of W is good, although it could be better controlled.

Therefore, as further work, other methods of adjusting W (like a specific-to-general policy) or maybe a stop criterion for the current method (leaving the value of W fixed after a certain point of the learning process) should be studied. Also, it could be interesting to extract other measures from the performance of the *GPM* tested, like the degree of diversity existing in the population.

Also, it would be very interesting to compare this bloat control method with recent Pareto-based Multi-Objective techniques like *MOLCS* [19]. This method is completely parameter-less, which was one of our goals. However, this means that the pressure applied to the complexity objective cannot be adjusted and, probably, it will be too strong or too mild for certain problems. If we know how the system will behave, having the possibility of some fine-tuning if quite desirable.

Acknowledgments

The authors acknowledge the support provided under grant numbers 2001FI 00514, TIC2002-04160-C02-02, TIC 2002-04036-C05-03 and 2002SGR 00155. Also, we would like to thank Enginyeria i Arquitectura La Salle for their support to our research group. The Wisconsin breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The primary tumor domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Slovenia. Thanks go to M. Zwitter and M. Soklic for providing the data.

References

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
2. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Company, Inc. (1989)
3. Smith, S.F.: Flexible learning of problem solving heuristics through adaptive search. In: Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Los Altos, CA, Morgan Kaufmann (1983) 421–425
4. Holland, J.H.: Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Machine learning, an artificial intelligence approach. Volume II. (1986) 593–623
5. DeJong, K.A., Spears, W.M.: Learning concept classification rules using genetic algorithms. Proceedings of the International Joint Conference on Artificial Intelligence (1991) 651–656
6. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175
7. Langdon, W.B.: Fitness causes bloat in variable size representations. Technical Report CSRP-97-14, University of Birmingham, School of Computer Science (1997) Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97.
8. Mitchell, T.M.: Machine Learning. McGraw-Hill (1997)
9. Bacardit, J., Garrell, J.M.: Métodos de generalización para sistemas clasificadores de Pittsburgh. In: Proceedings of the "Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)". (2002) 486–493
10. Rissanen, J.: Modeling by shortest data description. Automatica **vol. 14** (1978) 465–471
11. Pfahringer, B.: Practical uses of the minimum description length principle in inductive learning (1995)
12. Bacardit, J., Garrell, J.M.: Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system. In: Proceedings of the Genetic and Evolutionary Computation Conference - GECCO2003, (in press), LNCS, Springer (2003)
13. Gao, Q., Li, M., Vitányi, P.: Applying mdl to learn best model granularity. Artificial Intelligence **121** (2000) 1–29
14. Quinlan, J., Rivest, R.: Infering decision trees using the minimum description length principle. Information and Computation (1989) 227–248

15. Iba, H., de Garis, H., Sato, T.: Genetic programming using a minimum description length principle. In Kinnear, Jr., K.E., ed.: Advances in Genetic Programming. MIT Press (1994) 265–284
16. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
17. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference. (2002) 829–836
18. Bernadó, E., Mekaouche, A., Garrell, J.M.: A Study of a Genetic Classifier System Based on the Pittsburgh Approach on a Medical Domain. In: 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-99. (1999) 175–184
19. Llorà, X., Goldberg, D.E., Traus, I., Bernadó, E.: Accuracy, Parsimony, and Generality in Evolutionary Learning System a Multiobjective Selection. In: Advances in Learning Classifier Systems: proceedings of the 5th International Workshop on Learning Classifier Systems, (in press), LNAI, Springer (2002)
20. Bernadó, E., Garrell, J.M.: Multiobjective learning in a genetic classifier system (MOLeCS). Butlletí de l'Associació Catalana l'Intel.ligència Artificial **22** (2000) 102–111
21. Golobardes, E., Llorà, X., Garrell, J.M., Vernet, D., Bacardit, J.: Genetic classifier system as a heuristic weighting method for a case-based classifier system. Butlletí de l'Associació Catalana d'Intel.ligència Artificial **22** (2000) 132–141
22. Rivest, R.L.: Learning decision lists. Machine Learning **2** (1987) 229–246
23. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS) **8** (1998) 3–30
24. Cantú-Paz, E.: On random numbers and the performance of genetic algorithms. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers (2002) 311–318
25. Bacardit, J., Garrell, J.M.: Evolution of multi-adaptive discretization intervals for A rule-based genetic learning system. In: Proceedings of the Eighth Iberoamerican Conference on Artificial Intelligence, LNAI vol. 2527, Springer (2002) 350–360
26. Wolpert, D.H., Macready, W.G.: No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM (1995)
27. Brodley, C.: Addressing the selective superiority problem: Automatic algorithm /model class selection (1993)
28. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of Genetic Algorithms, Morgan Kaufmann (1991) 69–93
29. Llorà, X., Garrell, J.M.: Knowledge-independent data mining with fine-grained parallel evolutionary algorithms. In: Proceedings of the Third Genetic and Evolutionary Computation Conference, Morgan Kaufmann (2001) 461–468
30. Blake, C., Keogh, E., Merz, C.: Uci repository of machine learning databases (1998) (www.ics.uci.edu/mlearn/MLRepository.html).
31. Martínez Marroquín, E., Vos, C., et al.: Morphological analysis of mammary biopsy images. In: Proceedings of the IEEE International Conference on Image Processing. (1996) 943–947
32. Martí, J., Cufí, X., Regincós, J., et al.: Shape-based feature selection for microcalcification evaluation. In: Imaging Conference on Image Processing, 3338:1215-1224. (1998)
33. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: IJCAI. (1995) 1137–1145

34. Witten, I.H., Frank, E.: Data Mining: practical machine learning tools and techniques with java implementations. Morgan Kaufmann (2000)
35. Aha, D.W., Kibler, D.F., Albert, M.K.: Instance-based learning algorithms. Machine Learning **6** (1991) 37–66
36. Bernadó, E., Garrell, J.M.: Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. Special Issue of the Evolutionary Computation Journal on Learning Classifier Systems (in press) (2003)