Analysis and improvements of the Adaptive Discretization Intervals knowledge representation

Jaume Bacardit and Josep Maria Garrell

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

Abstract. In order to handle classification problems with real-valued attributes using discretization algorithms it is necessary to obtain a good and reduced set of cut points in order to learn successfully. In recent years a discretization-based knowledge representation called *Adaptive Discretization Intervals* has been developed that can use several discretizers at the same time and also combines adjacent cut points. In this paper we analyze its behavior in several aspects. From this analysis we propose some fixes and new operators that manage to improve the performance of the representation across a large set of domains.

1 Introduction

Genetic Algorithms (GA) [1] have been applied extensively in recent years to Classification and Machine Learning tasks [2–5]. The most popular family of systems performing these tasks is known as Learning Classifier Systems (LCS), having two main approaches: the Pittsburgh LCS [6] and the Michigan LCS [2].

The classical knowledge representations [6, 2] used in these systems are nominal. Therefore, a discretization algorithm is needed in order to handle problems with real-valued attributes with these representations, treating the resulting intervals as nominal values. The performance of these systems is tied to the right election of these intervals. A good discretization algorithm has to balance the loss of information intrinsic to this kind of process and generating a reasonable number of cut points, that is, a reasonable search space.

In recent years a discretization-based knowledge representation has been developed which can handle these issues, called *Adaptive Discretization Intervals (ADI) rule representation* [7]. This representation is used inside a Pittsburgh *LCS* and uses rules that contain intervals built joining together the low level intervals provided by the discretization algorithm, thus collapsing the search space when it is possible. Also, this representation can use several discretization algorithms at the same time allowing the system to choose the correct discretization for each problem and attribute.

In this paper we analyze the behavior of this representation from various points of view. This analysis identifies some problems that lead us to propose a new recombination operator and also a fix to another one. These improvements increase the ADI representation performance in most domains (even significantly sometimes based on statistical tests). Several combinations of the improvements proposed in this paper are tested, comparing them to the original configuration and also to two Machine Learning systems across several domains.

The paper is structured as follows. Section 2 describes the framework of the classifier system used for this paper and the *ADI* knowledge representation. Then, the ADI representation analysis and proposal of new operators is explained in section 3. Next, section 4 describes the test suite used in the comparison. The results obtained are summarized in section 5, section 6 discusses the conclusions and some further work and, finally, section 7 presents some related work.

2 Framework and ADI knowledge representation

The *LCS* used for this paper is called GAssist (*Genetic Algorithms based claS-SIfier sySTem*) [7], and is a Pittsburgh style classifier system descendant on *GABIL*. A detailed description of the system is found in a previous paper [8].

2.1 Adaptive Discretization Intervals (ADI) knowledge representation

This subsection describes the main characteristics of the *ADI* knowledge representation. For this paper we have used the second revision of the representation (named *ADI2* in previous work [7]).

The semantical structure of each rule in ADI is taken from **GABIL** [6]: Each rule consists of a condition part and a classification part: *condition* → *decision*. Each condition is a Conjunctive Normal Form (CNF) predicate defined as:

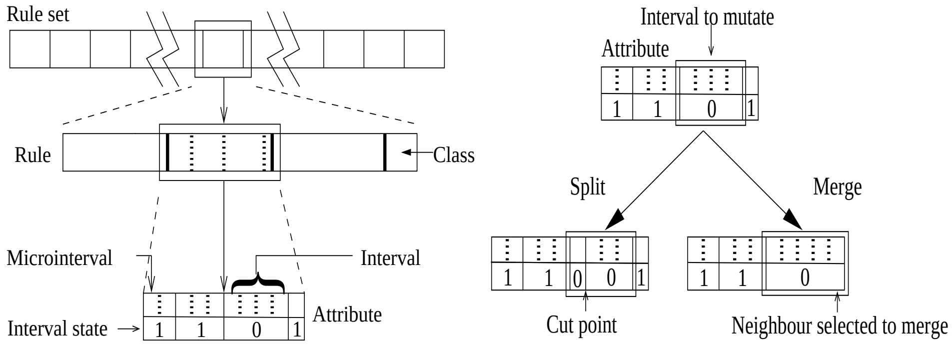
$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee \ldots A_n = V_m^b))$$

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 whose values can be {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.

In *GABIL* for each attribute we would use a set of static discretization intervals instead of nominal values. The intervals of the ADI 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 1, also showing the split and merge operations.

The inner workings of the representation and the exact definition of the operators can be read in [7].

Fig. 1. Adaptive intervals representation and the split and merge operators.



3 Analysis and improvements of the ADI knowledge representation

In this section we analyze the behavior of the *ADI* representation presented in previous papers and, after, we propose some fixes and a new operator in order to fix some identified flaws.

3.1 Discretizers in the population. Finding the ideal discretizer

The first issue we want to examine is the distribution of discretizers in the population. In the ADI representation, the initialization stage assigns a random discretizer from our predefined pool to each attribute term of each rule of each individual. Through the iterations, the discretizers of the best individuals survive, but no new discretizers are inserted into the population. This arises the question of how does the number of attributes in the population assigned to each discretizer evolve through the iterations. We extracted this information from the population and it is represented in figure 2. This figure show the evolution of the discretizer proportions for 4 problems (*bre,iris,mmg,pim*, detailed in section 4). The discretizers chosen for these tests are the most frequently used previously in the *ADI* research: uniform-width discretizer of 4,5,6,7,8,10,15,20,25 intervals. We show this figure to compare the behavior among the datasets. Therefore, the same Y scale is used in all plots.

Figure 2 shows that all discretizers start the evolutionary process with a proportion approximately of 1/number of discretizers. Later on, the proportions change through the iterations. We can see that the proportions for all the datasets do not diverge too much from their initial value, with the exception of the *iris* dataset. This behavior makes us wonder if the system has managed to identify the ideal discretizer for this dataset. Thus we repeated the tests for these four datasets but using only the discretizer most frequent for each problem. The results are detailed in table 1.

Fig. 2. Evolution of the discretizer proportions in the population for the *bre,iris,mmg,pim* datasets

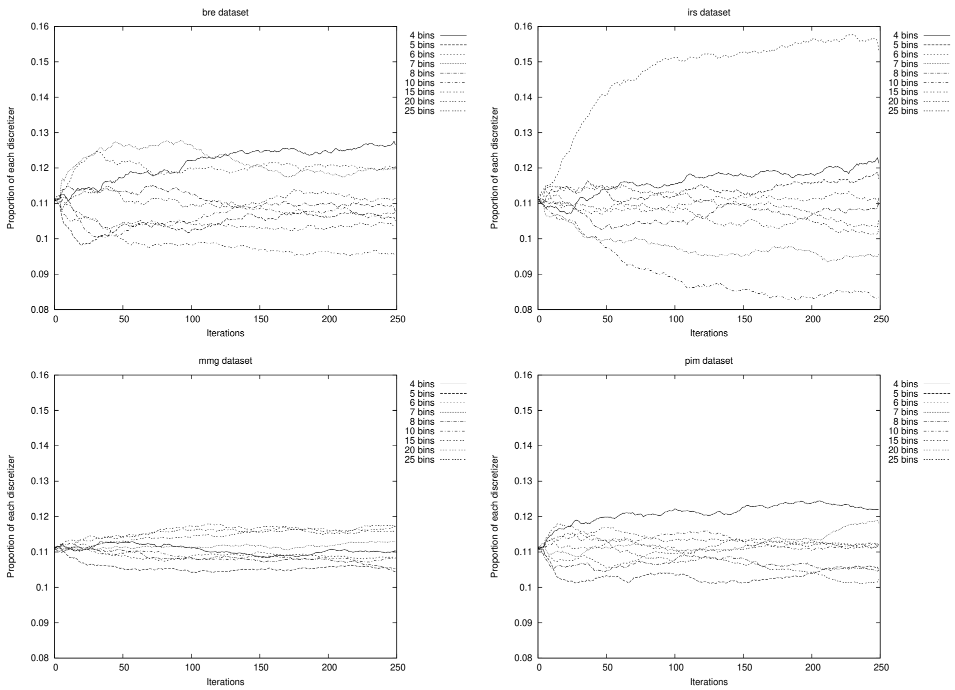


Table 1. Results of the experiment of using only the discretizer with more proportion in the population

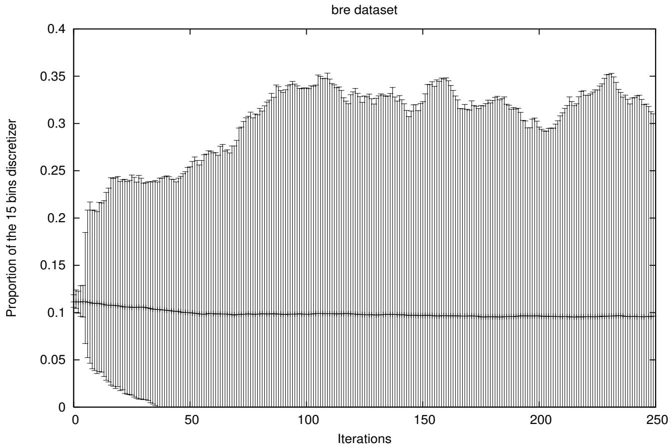
Dataset	Original accuracy	Accuracy with one discretizer	Discretizer
bre	95.6±2.2	95.5±1.8	4 intervals
irs	95.9±3.9	97.8±3.1	6 intervals
mmg	65.0±9.0	63.1±8.8	25 intervals
pim	74.4±4.7	74.2±3.7	4 intervals

We can see that the only dataset where there is accuracy increase when we are using only one discretizer is *iris*. It would be interesting to determine if there are other datasets where the evolution of the discretizer proportions presents the same behavior, and check if they manage also to identify the ideal discretizer. Unfortunately, we could not find any more dataset presenting this behavior.

3.2 Discretizers in the population. Survival of the discretizers

The next issue to analyze of the *ADI* representation is also related to the discretizer proportions in the population. In figure 3 we show the evolution of the average proportion of the 15 intervals discretizer for the *bre* dataset, but this time using error bars. We can extract an important observation: In some runs this discretizer disappeared from the population in less than 30 iterations. Other discretizers and datasets show the same behavior. Is this effect good or bad? Ideally the *GA* should choose the ideal discretizer for each domain and attribute. However, in most situations the system is not prepared to choose correctly in few iterations because it has not learned enough. It is clear that, in order to avoid this situation, we have to create some kind of mechanism that is able to introduce new discretizers into the population through the evolutionary process.

Fig. 3. Evolution of the proportions of the uniform-width discretizer with 15 intervals in the population for the bre dataset



What form does it take such survival mechanism? Probably the most suitable form would be an operator that changes the discretizer used by an attribute but maintaining, as much as possible, the semantical structure of the attribute. That is, finding a set of intervals build over the new discretizer as close as possible to the old ones. Unfortunately, an operator like this can present a huge computational cost considering that it is frequent to deal with datasets that have hundreds of cut points. Therefore, for this paper we have designed a simpler operator, called **reinitialize**. This operator repeats the process done in the initialization stage of the *GA*, but only for the selected individual and attribute term, as represented in figure 4.

Fig. 4. Steps of the reinitialize operator

1. We have an attribute term of a rule in the population selected for reinitialize
2. We select randomly a discretizer from our predefined pool
3. We assign a number of intervals to the attribute. This number is randomly chosen between 2 and min(number of *micro-intervals*, maximum allowed intervals per attribute)
4. We assign randomly a number of consecutive *micro-intervals* to each interval of the attribute. Every interval must have at least one *micro-interval*
5. We assign a random truth value (0 or 1) to each interval

This operator is applied after the merge and split stages, and the probability controlling it is also defined for each attribute-term. In order to assign a good value to this probability we did some tests with some probability values

(0.0025,0.005,0.01,0.015). We reproduce only the results for the *mmg* and *pim* datasets because they illustrate two different kinds of behavior. The results are in table 2, where we can see a correlation between the probability increase and the a decrease of obtained train accuracy and more rules and intervals per attribute. However, test accuracy does not show these trends. While the *pim* dataset does not benefit from this operator, the *mmg* dataset has a notable accuracy increase, considering that we are comparing two versions of the same system.

Table 2. Short tests of the reinitialize operator

Dataset	Reinit. prob.	Train acc.	Test acc.	# of rules	Interv. per attr
mmg	0.0000	78.4±1.8	65.0±9.0	6.6±1.0	2.4±0.1
	0.0025	78.4±1.8	65.7±8.8	6.5±1.0	2.5±0.1
	0.0050	78.2±1.7	66.8±8.8	6.5±0.8	2.5±0.1
	0.0100	77.5±1.7	67.3±9.5	6.5±0.9	2.6±0.1
	0.0150	76.4±1.8	67.5±9.1	6.6±1.0	2.7±0.1
pim	0.0000	78.6±1.0	74.4±4.7	5.4±0.9	2.2±0.1
	0.0025	78.1±0.9	74.4±4.5	5.1±0.6	2.2±0.1
	0.0050	78.1±1.0	74.4±4.7	5.3±0.7	2.3±0.1
	0.0100	77.9±1.0	74.3±4.3	5.2±0.8	2.3±0.1
	0.0150	77.7±1.0	74.1±5.1	5.1±0.6	2.4±0.1

Therefore, we can see that the operator is beneficial in some domains but its effects are too much aggressive (creating poor solutions) when applied to other datasets. Reinitialize needs to be redefined to have a softer behavior. The simplest way to achieve this goal is to redefine the probability controlling the operator. The new probability decreases linearly through the iterations until it achieves value 0 at last iteration. This fix allows the system to explore more aggressively in the early iterations and later on, in the final iterations, refine the good solutions. We repeated the short test with the same datasets, using as initial probabilities the values 0.01,0.02,0.03,0.04. Results are in table 3.

Table 3. Short tests of the improved reinitialize operator

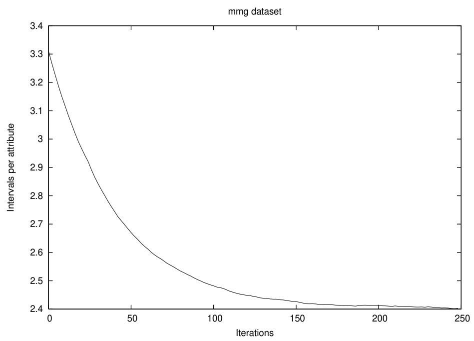
Dataset	Initial reinit. prob.	Train acc.	Test acc.	# of rules	Interv. per attr
mmg	0.00	78.4±1.8	65.0±9.0	6.6±1.0	2.4±0.1
	0.01	78.8±1.6	65.7±9.4	6.5±0.8	2.4±0.1
	0.02	78.4±1.7	66.2±9.4	6.5±1.0	2.5±0.1
	0.03	78.4±1.5	67.1±8.1	6.4±0.7	2.5±0.1
	0.04	78.0±1.8	67.2±8.0	6.6±1.0	2.5±0.1
pim	0.00	78.6±1.0	74.4±4.7	5.4±0.9	2.2±0.1
	0.01	78.7±1.0	74.6±4.5	5.3±0.9	2.3±0.1
	0.02	78.7±1.0	74.6±4.4	5.4±1.0	2.3±0.1
	0.03	78.6±1.0	75.1±4.5	5.3±0.7	2.3±0.1
	0.04	78.5±1.1	74.3±4.7	5.2±0.7	2.3±0.1

There are some interesting differences from the previous results. The train accuracy of tests with reinitialize operator is slightly higher that the original configuration. This shows that we have achieved the objective of creating an operator that helps exploring the search space for better solutions while being soft enough that it does not destroy these solutions in the final iterations. Even more interesting is the test accuracy, because now we obtain an accuracy increase over the original *ADI* configuration in both domains.

3.3 Evolution of the intervals per attribute ratio

Another important issue we want to analyze is how does it evolve the semantical structure of the rules through the iterations. That is, how many intervals per attribute do we have, and how are they distributed. We can see the evolution of the average number of intervals per attribute for the *mmg* problem in figure 5. All datasets present very similar behavior. The reason is the *hierarchical selection* operator [8] used to control the bloat effect. This operator promotes individuals that minimize the total sum of intervals of their rules, thus promoting individuals that have less rules and also less intervals per rule.

Fig. 5. Evolution of the average number of intervals per attribute



This behavior of the hierarchical selection is good for the system because it collapses the search space where it is possible and it helps creating generalized solutions that present low internal fragmentation of intervals. However, the difference between a well generalized attribute containing two intervals and an attribute too simple containing one (irrelevant) interval is only of one merge operation applied to the wrong attribute. In order to avoid part of this problem we introduce a **merge restriction** which cancels some merge operations if the attribute has only two intervals. The merge restriction process is represented in figure 6. The reader can see that now we have another probability to tune. For the sake of simplicity, in this paper we only test one value of this probability (0.5). The rationale of setting this value is to maintain an equilibrium between avoiding over-merging and creating too much specialized solutions.

Fig. 6. Merge operator with the new restriction included

1. An attribute has been selected for merge, according to the code in figure **??**.
2. If the attribute has only one interval → we cancel the operation
3. If the attribute has two intervals then
   (a) If random$[0, 1]$ < probability of merge restriction → we cancel the operation
   (b) Else → we apply the merge operation
4. If the attribute has more than two intervals → we apply the merge operation

4 Experimentation design

4.1 Test problems

For this paper we have selected 12 problems from the popular *UCI* repository [9] and also 3 real problems from private repositories (Biopsies [10], Mammograms [11] and Learning [7]). The characteristics of the problems are listed in table 4. The partition of the examples into the train and test sets was done using *stratified ten-fold cross-validation* [12].

Table 4. Characteristics of the test problems.

Name	ID.	Instances	Attributes	Classes
Bupa	bpa	345	6	2
Biopsies	bps	1027	24	2
Wisconsin Breast Cancer	bre	699	9	2
Glass	gls	214	9	6
Heart-Statlog	h-s	270	13	2
Ionosphere	ion	351	34	2
Learning	lrn	648	4	5
Mammograms	mmg	216	21	2
Pima-Indians-Diabetes	pim	768	8	2
Sonar	son	208	60	2
New-Thyroid	thy	215	5	3
Vehicle	veh	846	18	4
Wdbc	wdbc	569	30	2
Wine	wne	179	13	3
Wpbc	wpbc	198	33	2

4.2 Configurations of the GA to test

We tested 6 configurations of the *ADI* representation: the original version of the representation and 5 combination of the two improvements presented in this paper. The configurations are detailed in table 5. We have chosen to test the **reinitialize operator** always combined with the **merge restriction** because we think that the benefits of reinitialize diminish if we have too much generalization pressure. The GA parameters are shown in table 6. In order to have an external reference of the system performance, we have also included results for the *C4.5* [13] and IB1 [14] systems (using the *WEKA* [12] implementation and default parameters).

Table 5. Configurations of the *ADI* representation to test

Name	Merge restriction prob.	Reinitialize prob.	Reinitialize prob. at end
Original ADI	0	0	0
New ADI1	0.5	0	0
New ADI2	0.5	0.005	0.005
New ADI3	0.5	0.010	0.010
New ADI4	0.5	0.02	0
New ADI5	0.5	0.03	0

5 Results

In this section we present the results of the test described in the previous section. For each of the 6 *ADI* configuration and the 2 external systems we show the average and standard deviation of the cross-validation accuracy. The *ADI* results are the average of repeating the tests with 15 different random seeds.

Table 6. Common 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
Iterations	A maximum of 1500. Depends on learning rate on each dataset
Rule Deletion operator	
Iteration of activation	5
Minimum number of rules	number of classes of domain + 3
Hierarchical Selection	
Iteration of activation	25
Threshold	0.01
ADI rule representation	
Number of intervals of the uniform discretizations	4,5,6,7,8,10,15,20,25
Split probability	0.05
Merge probability	0.05

The results are presented in table 7. We applied two-sided Student t-tests to the results [12] with a significance level of 1%, in order to determine if there were significant outperformances between the methods tested. The results of the t-tests are shown in table 8.

Table 7. Mean and deviation of the accuracy for each method tested. Bold entries show the best method for each test problem

Prob.	Original ADI	New ADI1	New ADI2	New ADI3	New ADI4	New ADI5	C4.5	IB1
bpa	63.7±7.4	63.7±7.9	63.9±7.3	62.6±8.1	63.3±7.3	63.2±7.4	**68.4±3.9**	64.5±8.5
bps	80.6±4.3	80.7±4.0	80.7±3.8	79.6±4.1	80.6±4.3	80.0±4.0	80.1±4.5	**83.2±3.0**
bre	95.6±2.2	95.8±2.2	**96.0±2.1**	95.7±2.2	95.8±2.2	95.9±2.1	95.4±1.5	**96.0±1.4**
gls	66.4±7.7	66.5±7.8	**67.9±7.0**	**67.9±7.2**	67.8±8.1	66.5±7.8	65.8±9.9	66.3±10.4
h-s	80.4±6.9	80.2±6.7	**80.7±6.6**	79.8±7.2	80.5±6.7	80.6±7.1	76.3±5.5	74.1±6.4
ion	91.6±4.4	90.9±4.5	92.2±3.8	91.7±3.7	**92.7±3.6**	92.0±3.9	89.8±4.8	86.9±4.6
lrn	68.1±4.8	68.0±5.5	68.6±5.5	67.8±4.8	68.9±5.4	**69.0±4.7**	68.6±4.4	61.4±5.8
mmg	65.0±9.0	66.1±8.9	66.0±7.5	**67.8±9.4**	67.0±8.1	**67.8±8.6**	64.8±6.0	63.5±11.5
pim	74.4±4.7	75.1±4.3	74.7±4.0	74.3±4.8	**75.3±4.4**	74.4±4.5	73.1±5.0	70.3±3.3
son	74.6±9.7	73.2±10.1	73.1±9.6	72.3±10.6	74.3±9.2	73.5±9.5	71.5±8.0	**87.3±9.2**
thy	91.9±5.6	92.0±5.7	91.4±6.4	91.6±5.8	92.0±5.2	91.5±5.9	92.6±3.7	**96.8±4.7**
veh	66.0±4.9	66.4±4.9	66.1±4.7	65.6±4.0	66.7±4.4	66.5±4.7	**73.6±5.0**	69.4±5.0
wdbc	93.8±3.2	93.7±3.0	93.8±3.1	93.9±3.1	94.0±3.1	93.7±3.1	93.7±2.1	**95.6±2.1**
wine	92.7±6.9	92.5±6.9	92.2±6.9	92.6±6.7	93.0±6.8	92.2±6.9	94.1±6.8	**95.6±4.8**
wpbc	75.7±7.4	75.5±6.4	76.1±7.7	75.9±6.5	**77.1±6.4**	76.8±7.2	73.7±7.4	68.8±10.1
average	78.7±11.4	78.7±11.2	78.9±11.2	78.6±11.3	**79.3±11.2**	78.9±11.1	78.8±10.9	78.6±13.0

5.1 Analyzing only the performance of the ADI configurations

If we look only at the *ADI* configurations we can see that the original *ADI* configuration was only the best method in 1 of the 15 tested domains, showing that the improvements introduced to the representation perform well. Furthermore, in the domain (*son*) where the new operators performed worse, the difference was not significant (according to the t-tests). Looking at the accuracy averages and also to the t-tests, we can see that there is a clear winner: the *New ADI4* configuration. This method has the top accuracy average, it is the method that

Table 8. Summary of the statistical two-sided t-test performed at the 1% significance level. Each cell indicates how many times the method in the row outperforms the method in the column

Method	*Orig. ADI*	*New ADI1*	*New ADI2*	*New ADI3*	*New ADI4*	*New ADI5*	*C4.5*	*IB1*	*Total*
Orig. ADI	-	0	0	0	0	0	1	3	4
New ADI1	0	-	0	0	0	0	0	4	4
New ADI2	0	1	-	1	0	0	1	4	7
New ADI3	1	0	1	-	0	0	1	3	6
New ADI4	2	2	0	3	-	1	1	4	11
New ADI5	1	0	0	2	0	-	1	4	8
C4.5	2	1	1	2	2	2	-	0	10
IB1	3	4	4	3	3	3	2	-	22
Total	9	8	6	9	5	6	7	22	

outperforms significantly most often the other methods and, even more interesting, it has never been significantly outperformed.

Comparing *Original ADI* to *New ADI1* (where only the merge restriction improvement was used) we can see that they perform similarly in average, but *New ADI1* performs equal or better in the majority of domains. Thus, showing that it is a harmless fix that can be combined with the reinitialize operator without creating a bad interaction. The results of *New ADI3* show the dangers of the reinitialize operator where it is incorrectly tuned. It has the lowest accuracy average and it is the method most often significantly outperformed.

5.2 Comparing *ADI* to *C4.5* and *IB1*

If we now look at the results from a global point of view, we can see some big accuracy differences between the methods compared, specially between *IB1* and the other methods. These differences are quite logical, because they are a consequence of the *Selective Superiority Problem* [15]. Being *IB1* the only non axis-parallel classifier it was expected a change in the range of domains where this systems performs well.

Comparing the systems tested we can say that *New ADI4* is better (in average) than *C4.5* and *IB1*. Also, it is the system less times outperformed significantly, according to the T-Tests. This shows that this *ADI* configuration has a robust and reliable behavior.

6 Conclusions and further work

This paper was focused on an existing representation for real-valued attributes: the Adaptive Discretization Intervals (ADI) rule representation. This representation evolves rules that can use multiple discretizations, letting the evolution choose the correct discretization for each rule and attribute. Also, the intervals defined in each discretization can split or merge among them through the evolution process, reducing the search space where it is possible.

We have analyzed the behavior of the representation, studying its evolution through the iterations in two aspects: the proportions of each discretizer in the population and the number of intervals per attribute. From studying the proportions we have discovered that all discretizers can disappear sometimes from the population. Studying the evolution of the intervals per attribute ratio we have seen that the pressure applied to reduce this ratio can easily transform well generalized solutions into too much simple ones.

This analysis has led us to do two proposals: The first one is an operator that can reintroduce fresh discretizers and sets of intervals into the population. This operator, if badly tuned, can be very destructive. Thus, we have tested a dynamic probability decreasing its value through the iterations. The second one is a restriction introduced into the merge operator, in order to avoid the over-reduction of the number of intervals in the population.

We have tested 5 combinations of these two improvements with various parameterizations across 15 datasets and two alternative Machine Learning systems. The comparison of these configurations to the original *ADI* version shows that the improvements done have better performance. It is interesting to remark that the configuration performing better in average (*New ADI4*) has been outperformed significantly very few times, showing that the improvements presented here are good and robust.

As further work it would be interesting to combine these improvements with alternative sets of discretizers, containing both uniform and non-uniform discretization algorithms, as an alternative to the pool of uniform-width discretizers used for this paper. Also, we should analyze in depth the interactions between the two kind of improvements studied.

7 Related work

Discretization is not the only way to handle real-valued attributes in Evolutionary Computation based Machine Learning systems. Some examples are induction of decision trees (either axis-parallel or oblique), by either generating a full tree by means of genetic programming operators [3] or using an heuristic method to generate the tree and using a Genetic Algorithm and an Evolution Strategy to optimize the test performed at each node [16]. Other examples are inducing rules with real-valued intervals [17, 18] or generating an instance set used as the core of a *k-NN* classifier [3]. If our *ADI* method is able to find the correct cut points, the performance of the system should be quite competitive if compared to all the axis-parallel methods described above (all but the last one). Also, there are other systems that, like *ADI*, perform evolutionary induction of rules based on discretization [4, 5]. A comparison of *ADI* with these two methods is found in [19].

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.

References

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press (1975)
2. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175
3. 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
4. Giráldex, R., Aguilar-Ruiz, J., Riquelme, J.: Natural coding: A more efficient representation for evolutionary learning. In: GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference, Springer (2003) 979–990
5. Divina, F., Keijzer, M., Marchiori, E.: A method for handling numerical attributes in GA-based inductive concept learners. In: GECCO 2003: Proceedings of the Genetic and Evolutionary Computation Conference, Springer (2003) 898–908
6. 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
7. 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, LNCS 2724, Springer (2003) 1818–1831
8. Bacardit, J., Garrell, J.M.: Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In: Proceedings of the 6th International Workshop on Learning Classifier Systems, (in press), LNAI, Springer (2003)
9. Blake, C., Keogh, E., Merz, C.: UCI repository of machine learning databases (1998) (www.ics.uci.edu/mlearn/MLRepository.html).
10. 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
11. 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)
12. Witten, I.H., Frank, E.: Data Mining: practical machine learning tools and techniques with java implementations. Morgan Kaufmann (2000)
13. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
14. Aha, D.W., Kibler, D.F., Albert, M.K.: Instance-based learning algorithms. Machine Learning **6** (1991) 37–66
15. Brodley, C.: Addressing the selective superiority problem: Automatic algorithm /model class selection (1993)
16. Cantu-Paz, E., Kamath, C.: Inducing oblique decision trees with evolutionary algorithms. IEEE Transactions on Evolutionary Computation **7** (2003) 54–68
17. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In Booker, L., Forrest, S., Mitchell, M., Riolo, R.L., eds.: Festschrift in Honor of John H. Holland, Center for the Study of Complex Systems (1999) 111–121
18. Stone, C., Bull, L.: For real! xcs with continuous-valued inputs. Evolutionary Computation Journal **11** (2003) 298–336
19. Aguilar, J., Bacardit, J., Divina, F.: Experimental evaluation of discretization schemes for rule induction. In: GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference, Springer (to appear) (2004)