Incremental Learning for Pittsburgh Approach Classifier Systems

Jaume Bacardit and Josep Maria Garrell

Abstract— **One of the main problems of the Genetic Based Machine Learning (GBML) systems based on the Pittsburgh Approach has traditionally been the computational cost. One way to reduce this cost is to use techniques such as incremental learning or windowing, where a subset of the train examples are used for the learning process instead of the whole set. In this paper we present and analyse some methods that vary the subset of train examples used through the iterations. The results obtained show that beside the expected computational cost reduction these methods also achieve some accuracy gain.**

Key words—**Pittsburgh Approach Classifier Systems, Incremental Learning**

I. Introduction

THE application of Genetic Algorithms (GA) [1], [2] to classification problems is usually known as Genetic Based Machine Learning (GBML), and it has traditionally been addressed from two different points of view: the Pittsburgh approach, and the Michigan approach, early exemplified by LS-1 [3] and CS-1 [4], respectively.

One of the main drawbacks of the systems based on the Pittsburgh approach is the high computational cost. This cost is a consequence of the fact that each individual evolves a complete (and variable-length) solution of the classification problem. Thus, each fitness computation needs to classify all the train examples.

There are two ways to reduce this computational cost: (a) reduce the size of the individuals or (b) reduce the dimensionality of the problem we want to solve. The methods that use the former way are usually called parsimony or generalization pressure methods [5], [6]. The latter way is followed by methods that are usually identified as feature selection if they apply to problem attributes or incremental learning, windowing, prototype selection, etc. if they apply to train examples.

All the methods related to train examples share a common idea but applied from different points of view: the use of a subset of the train examples for the learning process. From a general Machine Learning point of view, we distribute these in three main categories:

Enginyeria i Arquitectura La Salle, Universitat Ramon Llull. Passeig Bonanova 8, 08022, Barcelona. E-mail: {jbacardit,josepmg}@salleURL.edu

- **Wrapper Methods** [7], [8], [9], [10], [11], [12]. These methods build a layer over the learning process which selects the correct subset of examples by running iteratively the unmodified learning algorithm. The subset of train examples used varies through the iterations until the stop criteria is achieved. This stop criteria usually is based on the estimation that the current subset of examples is similar enough to the whole set.
- **Modified learning algorithms** [8], [13], [14]. In this category we include the methods that have been modified to include the incremental learning inside their algorithm or methods that include/discard train examples based on knowledge-representation specific information.
- **Prototype Selection** [10], [15], [16]. In this category fall the methods that reduce the train set before the learning process. Thus, the learning is performed only once unlike the two prior categories. We want to remark that this definition is related to the structure of the incremental learning system, because the general definition of prototype selection is much broader than what we state here.

The wrapper methods are a very simple idea and easy to implement, but they are not a good choice if the computational cost of the chosen learning algorithm is considerable (like in our case).

If we focus specifically on Evolutionary Computation systems using incremental learning, Alex Freitas did a good classification of these systems [17] defining three types of incremental learning:

- Individual-wise: changing the subset of the train examples used for each fitness computation.
- Run-wise: selecting a static subset of examples for the whole evolutionary process.
- Generation-wise: changing the subset of the train examples used at each generation of the evolutionary process.

In this paper we present and analyse some incremental learning schemes that implement a generation-wise incremental learning using a modified learning algorithm. The differences between the schemes are the criteria used to change the selected subset of examples.

The test will show that the incremental learning schemes can reduce the computational cost of the system without accuracy loss. Even there is accuracy increase in most of the problems.

The paper is structured as follows: Section II presents some related work. Then, we describe the framework of our classifier system section III. The incremental learning schemes are presented in section IV. Next, section V describes the test suite used in the comparison. The results obtained are summarized in section VI. Finally, section VII discusses the conclusions and some further work.

II. Related work

There is plenty of related work about methods which reduce in some way the train examples used for the learning process, although there exist several names to identify these methods. Next we explain some related approaches for each category described in the introduction section.

A. Wrapper Methods

The first approach is the *batch-incremental* learning method used in *GABIL* [7], a Pittsburgh approach GBML system by De Jong and Spears. This system runs iteratively the batch learning algorithm *GABL* using growing sizes of train examples at each iteration. The system initially uses only one train example. *GABL* is run to generate a theory (rule sets, instance sets, etc). This theory is used to predict the classification of the next example. If this classification fails, *GABL* is run again with the two train examples. If not, the second example is simply stored in the train set and the prediction test is repeated for the rest of examples. After any incorrect prediction *GABL* is run again.

A generalized scheme of the above incremental learning method is defined by Fürnkranz with the name of *windowing algorithm* [8]. This scheme defines an initial train set size (*window*) and also a maximum incremental size, which defines the maximum number of misclassified examples which can be added to the window at each iteration of the scheme. The main difference from the above system is that the correctly classified examples are not added to the train set. The *C4.5* system [9] by Quinlan also follows this scheme for its windowing version.

All the previously described systems stop iterating when the train examples not included in the window can be classified correctly, using the theory generated in the current iteration. The *dynamic sampling* method [10] by John and Langley uses a different approach. This method estimates the accuracy of the whole set and stops adding more examples to the window when the difference between the accuracy of the current subset and this estimation falls below a certain threshold. After each iteration, the K examples used to calculate the current accuracy (acting as a test set) are added to the train set.

The three next approaches identify themselves as *prototype selection*, but they are included here instead of in the *prototype selection* category because the selection is achieved by iteratively running some learning algorithm, instead of using some other technique or heuristic.

First, two methods defined by Skalak [11] as *Sampling* and *Random Mutation Hill Climbing*. Sampling uses the *Monte Carlo* statistical tool to select n samples of m examples from the train set and runs the learning algorithm with each sample. The sample which produces the theory with best accuracy is selected. The second method defines a binary string with one bit for each train example which selects/unselects it and uses a local search method, the *Random Mutation Hill Climbing* to find a good string. That is, a good subset of examples.

A similar method was defined by Sierra et al. [12], but instead of *Random Mutation Hill Climbing* they use *Estimation of Distribution Algorithms*.

B. Modified Learning Algorithms

In this category we include the systems that either (a) have integrated the incremental part of the system inside the learning algorithm or (b) are wrapper methods which use knowledge representation specific information or the partial theory generated to add/discard examples from the train set for the next iteration.

We find with the name *Integrative Windowing*, developed by Fürnkranz [8], a wrapper method that differs from the general *Windowing* schema about how the final theory is generated. It is the union of the partials theories obtained at each iteration of the incremental learning. This approach is feasible because it uses separate-and-conquer learning algorithm and this makes easy to merge the partial theories. At the end of each iteration the train examples that are well covered by the accumulated theory are discarded, reducing the current window and the computational cost. The author also reports accuracy gain in the use of *Integrative Windowing* and discusses the effect of noise in incremental learning and proposes a noise-tolerant version of his algorithm (*Noise Tolerant Windowin*).

A similar system was proposed by Maloof and Michalski [13] named *Partial Memory Learning*. This method defines some refined policies to include and forget examples from the current window (partial memory). These policies use information from the knowledge representation to detect the relevant and irrelevant examples.

We also include in this category several methods in the *Case-Based Reasoning* and *Instance-Based Learning* fields, usually identified as *Case Base Reduction* techniques. An extensive review of these techniques can be found in [14].

C. Prototype selection

This category includes the methods which reduce the train set before the learning algorithm is run. Thus, the learning algorithms doesn't need to be modified and it is only run once.

A very simple approach in this category is called *static sampling* [10]. This method selects a sample of the train set and uses some statistical tests to determine if the sample is sufficiently similar to the whole train set. The χ^2 hypothesis test is used for categorical attributes and a large-sample test relying on the central limit theorem is used for numerical attributes.

Another approach which is suited for axis-parallel knowledge representations is *Editing by Ordered Projection* by Aguilar et al. [15]. It is an heuristic method based on a geometrical projection of the examples.

Finally, we include in this category the *Sort-Out Techniques* by Salamó and Golobardes [16] which uses the data-mining method *Rough Sets* to reduce a priori the *Case Base* for its application to *Case-Based Reasoning*.

III. Framework

In this section we describe the main features of our classifier system. GAssist (*Genetic Algorithms based claSSIfier sySTem*) [18] is a Pittsburgh style classifier system based on GABIL [7]. Directly from GABIL we have taken the knowledge representation for discrete attributes (rules with conjunctive normal form (CNF) predicates), the semantically correct crossover operator and the fitness computation (squared accuracy).

A. Matching strategy:

The matching process follows a "if ... then ... else if ... then..." structure, usually called *Decision List* [19].

B. 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 for mutation.

C. Control of the individuals length:

Dealing with variable-length individuals arises some serious considerations. One of the most important ones is the control of the size of the evolving individuals [5]. This control is achieved in GAssist using two different operators:

- *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 number of rules of an individual never goes below a threshold. This threshold is assigned to the number of classes of the domain.
- *Selection bias using the individual size*. Selection is guided as usual by the fitness (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 "size-based comparison" [6]. This operator considers two individuals "similar" if their fitness difference is below a certain threshold (d_{comp}). Then, it selects the individual with fewer number of rules. The tests in our previous paper showed that sizing d_{comp} to 0.01 was quite good for real problems and 0.001 was quite good for synthetic problems. Although a fine tuning of this parameter probably would improve the performance for each test domain, for the sake of simplicity we will use these values.

D. Knowledge Representation for real-valued attributes

In order to solve problems with real-valued attributes we use our own representation called *adaptive discretization intervals* rule representation [20], [21]. It is an evolution of the *GABIL* discrete rule representation where the discretization intervals can split and merge among them through the iterations. As the focus of this paper is not related to the knowledge representation, the details of the representation are omitted.

IV. Incremental Learning schemes

In this section we will present our proposal of incremental learning schemes. As was said previously in the introduction section, our schemes follow a simple idea: varying the subset of train examples used for the fitness computations through

```
Procedure Basic Incremental Learning
Input : Examples, NumSegments, NumIterations
Initialize GA
Reorder Examples in NumSegments parts of equal
class distribution
Iteration = 0
SegmentSize = size(Examples)/NumSegments
While  Iteration < NumIterations
    If  Iteration = NumIterations − 1 Then
      TrainSet = Examples
    Else
      CurrentSegment = int(Iteration · NumSegments
        /NumIterations)
      TrainSet= examples from
        Examples[CurrentSegment · SegmentSize] to
        Examples[(CurrentSegment + 1) · SegmentSize]
    EndIf
    Run one iteration of the GA with TrainSet
    Iteration = Iteration + 1
EndWhile
Output : Best set of rules from GA population
```

Fig. 1. Pseudocode of the Basic Incremental Learning scheme

TABLE I

QUICK TEST OF THE *BIL* SCHEME

Segments	Test accuracy	acc. increase	speedup
1	80.2%	—	—
2	79.5%	-0.7%	1.92
3	79.0%	-1.2%	2.96
4	78.9%	-1.3%	4.81

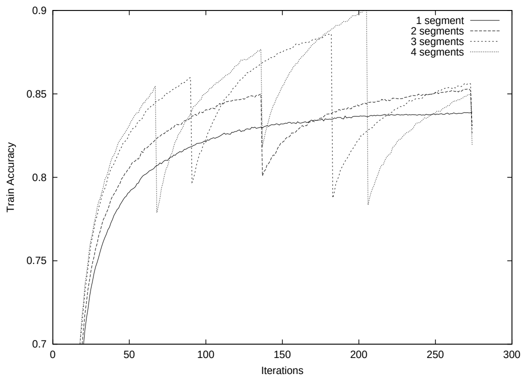

Fig. 2. Accuracy evolution for the Basic Incremental Learning scheme for the *Bps* problem

the *GA* iterations. The criteria to select this subset (*segment*) and also when and how often this segment is changed will define each one of the 3 schemes presented.

A. Basic Incremental Learning (BIL)

Our first proposal divides the train set and the *GA* iterations in N uniform segments, and uses the train segment 1 for the first segment of iterations, train segment 2 for the second segment of iterations, and so on. The last iteration of the learning process will use the whole train set because we need to select the individual which will provide the final theory, and it should be a good solution for all the train data, not only for the final segment. The original examples are reordered to maintain, for each segment, the same class distribution that exists in the whole train set. This scheme is represented by the pseudocode in figure 1

A quick test was done to check the performance of this *BIL* scheme. The test used the *Bps* problem which is detailed in section V. The test configuration and methodology are also detailed there. Tests with 2, 3 and 4 segments were done, and the results are shown in table I. The meaning of the column labeled *speedup* is the ratio between the non-incremental and incremental schemes run times. This speedup cannot be compared to the speedup measure of algorithm parallelization because we are modifying the algorithm. For this reason, we can find super-linear speedups.

The speedups obtained using 2 and 3 segments were as expected, and a surprise for the 4 segments test. However, there was an accuracy (percentage of correctly classified examples) loss for all the incremental tests. This results arrise a question: Is this decrease normal? Looking at the systems described in the related work section we can see reports of both performance increase and decrease. Thus, we have to explore why we have a performance decrease and whether it can be fixed. Next scheme will try to solve this problem.

B. Basic Incremental Learning with a Total Segment (BILTS)

The reason of the performance decrease of the *BIL* scheme can be seen by looking at the accuracy evolution through the iterations. This evolution is represented in figure 2. At each segment change, there is an accuracy loss because the knowledge of the current population is not enough to classify some of the new train examples.

Thus, New knowledge has to be learned. The process of adding new knowledge will modify some of the current rules, but it will also add new ones. This can lead to a situation where some of the good rules for the previous segments are not used anymore, and these rules are deleted by our *rule deletion* operator. The old useless knowledge is forgotten. Looking again at the graph in figure 2 we can see a performance hit at the end of the learning. It is due to using the whole train segment in the last iteration. All the forgotten knowledge would be useful again, but it is missing.

How can we fix the problem of forgetting good knowledge and, as a consequence, prevent the performance loss? Switching off the rule deletion

```
Procedure Basic Incremental Learning with Total
            Segment
Input : Examples, NumSegments, NumIterations
Initialize GA
Reorder Examples in NumSegments parts of equal
class distribution
Iteration = 0
SegmentSize = size(Examples)/NumSegments
While Iteration < NumIterations
    CurrentSegment = int(Iteration·
        (NumSegments + 1)/NumIterations)
    If CurrentSegment = NumSegments Then
        TrainSet = Examples
    Else
        TrainSet= examples from
            Examples[CurrentSegment·
                        SegmentSize] to
            Examples[(CurrentSegment + 1)·
                        SegmentSize]
    EndIf
    Run one iteration of the GA with TrainSet
    Iteration = Iteration + 1
EndWhile
Output : Best set of rules from GA population
```

Fig. 3. Pseudocode of the Basic Incremental Learning with Total Segment scheme

TABLE II
PERFORMANCE OF THE *Basic Incremental Learning with Total Segment* SCHEME

Segments	Test accuracy	acc. increase	speedup
1	80.2%	—	—
2	80.1%	-0.1%	1.50
3	80.0%	-0.2%	2.06
4	80.5%	+0.3%	2.93

operator is not a feasible solution because the population would grow without control, as was seen in our previous work [6]. A simple alternative is adding a stage at the end of the learning process where all the train examples are used. This scheme, called *Basic Incremental Learning with a Total Segment (BILTS)* is represented by the pseudocode in figure 3.

We repeated the same quick tests done for the previous scheme, and its results can be seen in table II. Figure 4 shows the graph of the accuracy evolution. The performance loss is smaller now and even there is performance increase using 4 segments. The speedup, obviously, is less than the obtained with the previous scheme, but it is still worthy.

With this second scheme we have fixed the loss of knowledge derived from a segment change, but it has costed a decrease in the speedup of the system. Instead of fixing the loss of knowledge maybe we should prevent it and, therefore, avoid the stage of the learning using the whole train set.

C. Incremental Learning with Alternating Segments (ILAS)

In order to prevent a knowledge loss we should use all the train examples with enough frequency

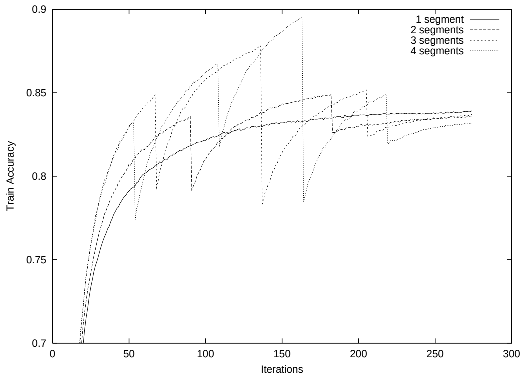

Fig. 4. Accuracy evolution for the Basic Incremental Learning with Total Segment scheme for the *Bps* problem

```
Procedure Incremental Learning with Alternating
            Segments
Input : Examples, NumSegments, NumIterations
Initialize GA
Reorder Examples in NumSegments parts of equal
class distribution
Iteration = 0
SegmentSize = size(Examples)/NumSegments
While Iteration < NumIterations
    If Iteration = NumIterations − 1 Then
        TrainSet = Examples
    Else
        CurrentSegment =Iteration mod
                            NumSegments
        TrainSet= examples from
            Examples[CurrentSegment·
                        SegmentSize] to
            Examples[(CurrentSegment + 1)·
                        SegmentSize]
    EndIf
    Run one iteration of the GA with TrainSet
    Iteration = Iteration + 1
EndWhile
Output : Best set of rules from GA population
```

Fig. 5. Pseudocode of the Incremental Learning with Alternating Segments scheme

to assure that all the knowledge of the individuals becomes useful and, as a consequence, it is not forgotten. Bringing this idea to the extreme, we propose another scheme called *Incremental Learning with Alternating Segments*, which changes the used segment at each iteration. Its pseudocode can be seen in figure 5.

Again, we repeated the same quick test. Its results can be seen in table III and in figure 6. Some oscillations have appeared in the evolution of the incremental schemes. They are due to the fact that there are some segments which are easier than some others. By easier segments we understand segments that have, by chance, less noise, more uniform distribution of examples, etc. Now the results are really surprising for both the accuracy increase and the speedup. How can we explain the performance and speedup increase?

Our hypothesis is based on the behavior of

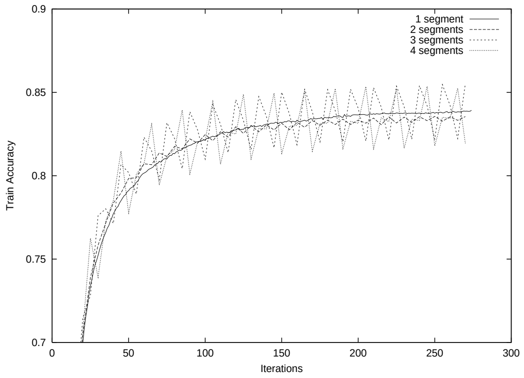

Fig. 6. Accuracy evolution for the Incremental Learning with Alternating Segments scheme for the *Bps* problem. Sampling of the accuracy every 5 iterations

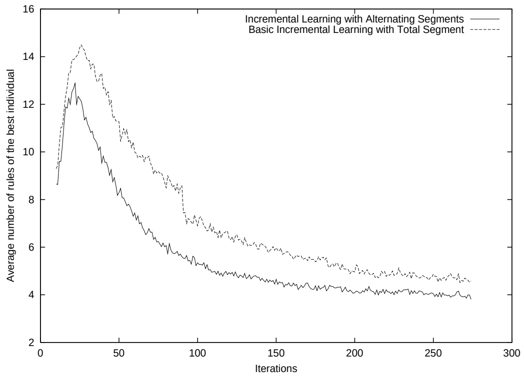

Fig. 7. Evolution of the number of rules for the incremental learning schemes with two segments in the *Bps* problem

TABLE III

PERFORMANCE OF THE *Incremental Learning with Alternating Segment* SCHEME

Segments	Test accuracy	acc. increase	speedup
1	80.2%	—	—
2	80.5%	+0.3%	2.05
3	80.8%	+0.5%	3.6
4	80.4%	+0.2%	6.01

the classifier system given our fitness function (squared train accuracy). This fitness function does not make differences whether train examples are learned (creating generalized rules) or memorized (creating specific rules). Using a non-incremental *GA*, the easiest way to increase the fitness is to memorize examples instead of learning them. Generality pressure methods can reduce this problem, but not completely fix it.

The *ILAS* scheme changes dramatically the environment of the GA population: The adaptation of the individuals to the high frequency changes of the train set consists of learning the examples instead of memorizing them, because the chances of surviving in the population become higher. The use of the rule deletion operator implicates that all the rules have to, at least, match one example in every segment. This fact penalizes specific rules.

Therefore, more generalized solutions are generated and these solutions have more chances of having a good test accuracy. Also, more generalized solutions usually mean smaller solutions. This is the reason of the speedup increase and can be seen in figure 7.

V. TEST SUITE

In the previous section we have presented three incremental learning schemes, and we have shown the results of some quick tests using only one dataset. In this section we summarize the results of a more extensive experimentation using problems with very diverse number of attributes and sizes. These experiments will be used to to check what problems can be solved successfully using incremental learning schemes, and also which degree of incrementality can we reach successfully.

A. Test problems

First, we use three synthetic problems:
- *tao* [22]. A problem that has non-orthogonal class boundaries and represents the tao figure.
- *mux* [23]. The 11 input multiplexer.
- *lex* [24]. The binary representation of a seven-segments display using the represented digit as the class. This problem comes from University of California at Irvine (UCI) repository and has an added artificial 10% degree of noise.

We use several real problems also provided by the UCI repository. The selected problems are:
- Pima-indians-diabetes (*pim*)
- Iris *irs*
- Glass *gls*
- Winsconsin Breast Cancer (*bre*)
- Ionosphere *ion*

Finally we will use three problems from our own private repository:
- *bps* [25]. This problem deals with the diagnosis of breast cancer based on biopsies.
- *mmg* [26]. Another problem of breast cancer diagnosis, this time based on mammograms.
- *lrn* [18]. This problem is related to the prediction of student qualifications.

The characteristics of the problems are listed in table IV, and the partition of the examples into the train and test sets was done using the *stratified ten-fold cross-validation* method [27].

B. Type of tests and configuration of the GA

In these extensive tests we have selected the two last incremental learning schemes described in the previous section. The *BIL* scheme was discarded from these tests because the second

TABLE IV

CHARACTERISTICS OF THE TEST PROBLEMS.

Dataset	#examples	real attr.	discrete attr.	classes
tao	1888	2	-	2
mux	2048	-	11	2
led	2000	-	7	10
pim	768	8	-	2
irs	150	4	-	3
gls	214	9	-	6
bre	699	-	9	2
ion	351	34	-	2
bps	1027	24	-	2
mmg	216	21	-	2
lrn	648	4	2	5

TABLE V

COMMON AND PROBLEM-SPECIFIC PARAMETERS OF THE GA.

Parameter	Value
Crossover probability	0.6
Crossover operator	1px
Iter. of rule eliminating activation	10
Iter. of size-based comparison activation	10
Tournament size	3
Population size	300
Probability of mutating an individual	0.6
Numbers of incremental learning segments	2,3,4

Code	Parameter
#iter	Number of GA iterations
d_{comp}	Distance param. in “size-based comparison” op.

Problem	Parameter	
	#iter	d_{comp}
tao	900	0.001
mx11	450	0.001
led	1000	0.001
pima	250	0.01
iris	250	0.01
glass	1100	0.01
breast	250	0.01
ionosphere	450	0.01
bps	275	0.01
mamm	250	0.01
lrn	700	0.01

scheme (*BILTS*) solves its weak point. For each incremental scheme we run tests with 2, 3 and 4 segments. As a baseline we include the non-incremental scheme (*NON*).

The pseudo-random number generator (PRNG) used is the *Mersene Twister* [28], which is one of the best PRNGs currently available. Its application to *GAs* has been studied recently [29] showing that it reduces the fluctuation of the obtained results. The seed used to initialize the PRNG are taken from pure random data downloaded from www.random.org. Each test was repeated with 15 different seeds and the results obtained were averaged. The GA parameters used are shown in table V.

VI. RESULTS

This section presents the results obtained. The aim of the tests was to compare the methods proposed among them in this paper in two aspects: accuracy and computational cost. Therefore, for each method and test problem we show the averaged cross-validation accuracy and also the accuracy and speedup increase over the non-incremental test.

Results are shown in table VI and are summarized using the ranking in table VII. The ranking for each problem and method is based on the accuracy. The global rankings are computed averaging the problem rankings.

An statistical Student t-test was applied to the results to determine if any of the incremental methods outperformed or was outperformed by the non-incremental configuration. The results of the two-sided t-test using a significance level of 0.01 showed that the incremental methods were never significantly outperformed. On the other hand, the configurations *BILTS*, *ILAS2* and *ILAS3* for the *Ionosphere* problem managed to significantly outperform the non-incremental configuration.

What can we observe in the results? First of all, we can see that our original objective of computational cost reduction has been achieved for all the problems. Nevertheless, more interesting is the observation that the accuracy increase of the incremental learning schemes shown in section IV for the *bps* problem also appears for most of the problems tested. Specially, the results are surprising for the *ion* problem where both the accuracy increase and the speedup are substantial.

At first sight, it seems that there is no correlation between the speedup and the number of instances of each problem. A study of the degree of noise and redundancy in each domain would probably explain better the computational cost reduction on each problem.

Looking at the ranking table we can see that the best incremental learning scheme is the *Incremental Learning with Alternating Segments* using two segments. This configuration, with the exception of the first problem, always increases the accuracy over the non-incremental scheme, and its speedup ranges from 1.59 in the *led* problem to 3.19 in *ion*. Therefore, it seems a safe bet as a default incremental learning configuration.

Another interesting observation is the results obtained in the synthetic problems tests. The tests for the two problems without noise and redundant information (*tao* and *mux*) had very few or no improvement over the non-incremental scheme, while most of the tests for the problem with redundancy and artificially added noise (*led*) achieved some improvement. The improvement for the noisy domain tests shows that the incremental learning scheme is prepared to handle problems of this kind. On the other hand, the results for the domains without noise and re-

TABLE VI

Results of the tests for the *NON*, *BILTS* and *ILAS* schemes. Bold entries show the best configuration for each problem

Problem	scheme	accuracy	acc. increase	speedup
tao	NON	94.8%	—	—
	BILTS2	**95.0%**	+0.2%	1.37
	BILTS3	94.8%	+0.0%	1.79
	BILTS4	94.8%	+0.0%	2.27
	ILAS2	94.7%	-0.1%	1.94
	ILAS3	94.6%	-0.2%	2.91
	ILAS4	94.3%	-0.5%	4.04
mux	NON	**100.0%**	—	—
	BILTS2	99.9%	-0.1%	1.46
	BILTS3	99.9%	-0.1%	2.05
	BILTS4	99.9%	-0.1%	2.73
	ILAS2	**100.0%**	+0.0%	1.76
	ILAS3	99.9%	-0.1%	3.14
	ILAS4	99.9%	-0.1%	4.63
led	NON	74.3%	—	—
	BILTS2	74.4%	+0.1%	1.43
	BILTS3	74.4%	+0.1%	1.90
	BILTS4	74.2%	-0.1%	2.37
	ILAS2	**74.5%**	+0.2%	1.59
	ILAS3	**74.5%**	+0.2%	2.46
	ILAS4	**74.5%**	+0.2%	3.27
pim	NON	74.3%	—	—
	BILTS2	74.4%	+0.1%	1.67
	BILTS3	74.4%	+0.1%	2.23
	BILTS4	**74.6%**	+0.3%	2.82
	ILAS2	74.5%	+0.2%	2.41
	ILAS3	74.0%	-0.3%	3.76
	ILAS4	73.7%	-0.6%	5.08
irs	NON	95.4%	—	—
	BILTS2	95.8%	+0.4%	1.39
	BILTS3	95.6%	+0.2%	1.62
	BILTS4	95.9%	+0.5%	1.85
	ILAS2	96.1%	+0.7%	1.67
	ILAS3	**96.2%**	+0.8%	2.14
	ILAS4	95.9%	+0.5%	2.49
gls	NON	65.1%	—	—
	BILTS2	65.8%	+0.8%	1.29
	BILTS3	**66.4%**	+1.3%	1.61
	BILTS4	64.5%	-0.6%	1.83
	ILAS2	65.6%	+0.5%	1.61
	ILAS3	65.2%	+0.1%	2.11
	ILAS4	64.6%	-0.5%	2.41
bre	NON	95.8%	—	—
	BILTS2	95.7%	-0.1%	1.74
	BILTS3	95.8%	+0.0%	2.54
	BILTS4	95.5%	-0.3%	3.02
	ILAS2	**95.9%**	+0.1%	2.71
	ILAS3	95.8%	+0.0%	4.76
	ILAS4	95.8%	+0.0%	5.91
ion	NON	84.5%	—	—
	BILTS2	85.4%	+0.9%	1.61
	BILTS3	88.0%	+3.5%	2.66
	BILTS4	86.1%	+1.6%	3.06
	ILAS2	**89.0%**	+4.5%	3.19
	ILAS3	**89.0%**	+4.5%	6.60
	ILAS4	87.1%	+2.6%	8.55
bps	NON	80.2%	—	—
	BILTS2	80.1%	-0.1%	1.50
	BILTS3	80.0%	-0.2%	2.06
	BILTS4	80.5%	+0.3%	2.93
	ILAS2	80.5%	+0.3%	2.05
	ILAS3	**80.8%**	+0.5%	3.6
	ILAS4	80.4%	+0.2%	6.01
mmg	NON	64.8%	—	—
	BILTS2	63.8%	-1.0%	1.40
	BILTS3	**65.0%**	+0.2%	1.71
	BILTS4	64.8%	+0.0%	2.06
	ILAS2	**65.0%**	+0.2%	1.76
	ILAS3	64.5%	-0.3%	2.36
	ILAS4	64.1%	-0.7%	2.87
lrn	NON	66.6%	—	—
	BILTS2	66.5%	-0.1%	1.45
	BILTS3	67.2%	+0.6%	2.08
	BILTS4	66.6%	+0.0%	2.32
	ILAS2	**67.4%**	+0.8%	2.26
	ILAS3	67.2%	+0.6%	3.51
	ILAS4	66.4%	-0.2%	4.68

TABLE VII

Performance ranking of the tested methods. Lower average means better ranking.

Probl.	NON	BILTS2	BILTS3	BILTS4	**ILAS2**	ILAS3	ILAS4
tao	2	**1**	2	2	5	6	7
mux	**1**	3	3	3	**1**	3	3
led	6	4	4	7	**1**	**1**	**1**
pim	5	3	3	**1**	2	6	7
irs	7	5	6	3	2	**1**	3
gls	5	2	**1**	7	3	4	6
bre	2	6	2	7	**1**	2	2
ion	7	6	3	5	**1**	**1**	4
bps	5	6	7	2	2	**1**	4
mmg	3	7	**1**	3	**1**	5	6
lrn	4	6	2	4	**1**	2	7
Ave.	4.27	4.45	3.09	4	**1.82**	2.91	4.55
Rank	5	6	3	4	**1**	2	7

dundant information show that the incremental learning scheme should be given more iterations to reach the same learning rate in this kind of problems.

VII. Conclusions and further work

In this paper we have introduced and tested some incremental learning methods applied to Pittsburgh approach Classifier Systems. The objective of these methods is to reduce the cost of fitness computations by using only a subset of the train examples to evaluate each individual and, therefore, reducing the total computational cost of the system.

These methods introduce very simple modifications over the non-incremental learning scheme: varying the subset of used train examples over the iterations. Despite this simplicity, the tests performed show that the incremental learning schemes manage to reduce the computational cost for all the test problems while increasing or maintaining the accuracy obtained by the non-incremental system.

Comparing the results for the different incremental learning methods tested, we can conclude that the *Incremental Learning with Alternating Segments* algorithm, using two segments, is a good choice as the default incremental learning configuration.

As a further work, we should study the way to fix the weakness of the incremental learning schemes observed for the synthetic problems. As we have said before, the fix probably could be as simple as increasing the number of iterations of the *GA*. Therefore we should find a good balance between speedup and quality of the generated solutions.

Also, the motivation of the studying incremental learning schemes was to achieve a reduction of the computational cost of the system without accuracy loss. We have managed to outreach this objective by achieving accuracy increases. Therefore, we should study in more depth the learning capacity of these schemes.

Finally, we could to test if it is viable to use some of the incremental learning methods described in the related work section in combination with a GBML system and also other ways of creating and changing the segments.

Acknowledgments

The authors acknowledge the support provided under grant numbers 2001FI 00514, CICYT Tel08-0408-02, FIS 00/0033-02 and TIC 2002-04036-C05-03. The results of this work were partially obtained using equipment cofunded by the *Direcció General de Recerca de la Generalitat de Catalunya (D.O.G.C. 30/12/1997)*. Finally we would like to thank Enginyeria i Arquitectura La Salle for their support to our research group.

References

[1] John H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[2] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

[3] Stephen F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, Los Altos, CA, 1983, pp. 421–425, Morgan Kaufmann.

[4] John H. Holland, "Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," in *Machine learning, an artificial intelligence approach. Volume II*, pp. 593–623. 1986.

[5] Terence Soule and James A. Foster, "Effects of code growth and parsimony pressure on populations in genetic programming," *Evolutionary Computation*, vol. 6, no. 4, pp. 293–309, Winter 1998.

[6] Jaume Bacardit and Josep M. Garrell, "Métodos de generalización para sistemas clasificadores de Pittsburgh," in *Proceedings of the "Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)"*, 2002, pp. 486–493.

[7] Kenneth A. DeJong, William M. Spears, and Diana F. Gordon, "Using genetic algorithms for concept learning," *Machine Learning*, vol. 13, no. 2/3, pp. 161–188, 1993.

[8] Johannes Fürnkranz, "Integrative windowing," *Journal of Artificial Intelligence Research*, vol. 8, pp. 129–164, 1998.

[9] J. Ross Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[10] George H. John and Pat Langley, "Static versus dynamic sampling for data mining," in *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining, KDD*, Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, Eds. 2–4 1996, pp. 367–370, AAAI Press.

[11] David B. Skalak, "Prototype and feature selection by sampling and random mutation hill climbing algorithms," in *International Conference on Machine Learning*, 1994, pp. 293–301.

[12] B. Sierra, E. Lazkano, I. Inza, M. Merino, P. Larrañaga, and J. Quiroga, "Prototype selection and feature subset selection by estimation of distribution algorithms. A case study in the survival of cirrhotic patients treated with TIPS," *Lecture Notes in Computer Science*, vol. 2101, pp. 20–29, 2001.

[13] Marcus A. Maloof and Ryszard S. Michalski, "Selecting examples for partial memory learning," *Machine Learning*, vol. 41, no. 1, pp. 27–52, 2000.

[14] D. Randall Wilson and Tony R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.

[15] Jesus S. Aguilar-Ruiz, J. Riquelme, and M. Toro, "Data set editing by ordered projection," in *Proceedings of the European Conference on Artificial Intelligence*, 2000, pp. 251–255.

[16] M. Salamó and E. Golobardes, "Deleting and Building Sort Out Techniques for Case Base Maintenance," in *Advances in Case-Based Reasoning. 6th. European Conference on Case-Based Reasoning*, 2002, pp. 365–379.

[17] Alex A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, Springer-Verlag, 2002.

[18] Elisabet Golobardes, Xavier Llorà, Josep Maria Garrell, David Vernet, and Jaume Bacardit, "Genetic classifier system as a heuristic weighting method for a case-based classifier system," *Butlletí de l'Associació Catalana d'Intel.ligència Artificial*, vol. 22, pp. 132–141, 2000.

[19] Ronald L. Rivest, "Learning decision lists," *Machine Learning*, vol. 2, no. 3, pp. 229–246, 1987.

[20] Jaume Bacardit and Josep M. Garrell, "Evolution of adaptive discretization intervals for A rule-based genetic learning system," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, p. 677.

[21] Jaume Bacardit and Josep M. Garrell, "Evolution of multi-adaptive discretization intervals for A rule-based genetic learning system," in *Proceedings of the VIII Iberoamerican Conference on Artificial Intelligence (IBERAMIA'2002)*. 2002, pp. 350–360, LNAI vol. 2527, Springer.

[22] Xavier Llorà and Josep M. Garrell, "Knowledge-independent data mining with fine-grained parallel evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. 2001, pp. 461–468, Morgan Kaufmann.

[23] Steward W. Wilson, "Classifier fitness based on accuracy.," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[24] C. Blake, E. Keogh, and C. Merz, "Uci repository of machine learning databases," 1998, Blake, C., Keogh, E., & Merz, C.J. (1998). UCI repository of machine learning databases (www.ics.uci.edu/mlearn/MLRepository.html).

[25] E. Martínez Marroquín, C. Vos, and et al., "Morphological analysis of mammary biopsy images," in *Proceedings of the IEEE International Conference on Image Processing (ICIP'96)*, 1996, pp. 943–947.

[26] J. Martí, X. Cufí, J. Regincós, and et al., "Shape-based feature selection for microcalcification evaluation," in *Imaging Conference on Image Processing, 3338:1215-1224*, 1998.

[27] Ron Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, 1995, pp. 1137–1145.

[28] Makoto Matsumoto and Takuji Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[29] Erick Cantú-Paz, "On random numbers and the performance of genetic algorithms," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pp. 311–318.