Analysis of the Initialization Stage of a Pittsburgh Approach Learning Classifier System

Jaume Bacardit
Automated Scheduling, Optimisation and Planning group
School of Computer Science and IT, University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
jqb@cs.nott.ac.uk

ABSTRACT

This paper is focused on studying the initialization stage of learning classifier systems (LCS) applying the Pittsburgh approach. It has a theoretical part where the covering probability of a random rule set is modelled and a practical part. The practical part has the objective of developing general initialization policies that have competent performance on a broad range of datasets. Two kinds of policies are tested: (1) ways of tuning the initialization probability of the system and (2) smart initialization operators that create rules that are generalized versions of randomly sampled training instances. The results identify a subset of settings that are robust enough to be considered candidates to be the default initialization policy. These settings have competent performance compared to several alternative machine learning systems. Beside identifying the good policies, the experimentation made is also useful to give hints about what kind of initial solutions is the system able to process successfully to create well generalized solutions

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning

General Terms

Algorithms,Experimentation

Keywords

Evolutionary Algorithms, Learning Classifier Systems, Rule Induction

1. INTRODUCTION

In the last decade the behaviour of genetic algorithms (GA) [13, 11] has been analyzed using several different techniques and approaches. One of them proposes a facet-wise analysis [12] of selectorecombinative GAs based on the *building block* (BB) concept, proposing a set of models that are used to guarantee the success of a GA for problems of bounded difficulty.

More recently this analysis has been extrapolated to Learning Classifier Systems (LCS), one of the application of evolutionary computation to machine learning tasks. Specifically there is extensive theoretical work [8] to model the behaviour of *XCS* [19], one of the most representative systems of the Michigan approach of *LCS*.

The aim of this paper is to analyze and improve the initialization stage of one of the other major LCS families: the Pittsburgh approach [10]. This study is splitted in two parts. In the first part one of the theoretical models proposed for *XCS* is adapted for a modern-day Pittsburgh system: *GAssist* [3]. Specifically, the model studied in this paper deals with the supply of raw building blocks in the population. That is, the initialization stage of the system.

This theoretical model deals with the probability of matching a random input instance for the *GABIL* [10] nominal knowledge representation. The model is also applicable to representations that inherit the semantics of *GABIL*, such as the *Adaptive Discretization Intervals (ADI)* [3] knowledge representation for real-valued attributes.

In the second part of the paper a form of *smart* initialization operator is studied. This operator creates the initial rules of the population as generalized versions of randomly chosen instances from the training set, in a similar manner to the *covering* operator of *XCS* [19] or the initialization stage of the *HIDER* genetic iterative rule learning system [1]. Two instance sampling methods are studied.

These studied models and techniques are tested experimentally on a broad range of real datasets with one general goal: determining if there is a global initialization policy with competent performance. If this goal is achieved, it can alleviate the practitioner of the system from manually tuning the initialization stage of the system for each dataset. The performance of the system is also compared to a broad range of alternative learning systems comprising different learning paradigms and knowledge representations.

The paper is structured as follows. First, section 2 presents some related work, followed by section 3 describing briefly the framework of the classifier system studied in the paper. Then, sections 4 and 5 define the match probability model and the smart initialization operator, respectively. Next, section 6 describes the experimental evaluation reported in this paper. The results of this evaluation are

GECCO'05, June 25–29, 2005, Washington, DC, USA.

discussed in section 7 and, finally, section 8 describes the conclusions and proposes some further work.

2. RELATED WORK

There is extensive theoretical work [8] of the behaviour of *Michigan* style systems, specifically on *XCS*. Its behaviour is modelled from several different points of view: covering bound (to make sure that the initial rules can match the input instances), schema bound (to make sure that these initial rules contain good classifiers), reproductive opportunity bound (to ensure reproduction and growth of good rules), time bound (number of needed learning steps to achieve perfect accuracy), and niche support bound (ensure that all sub-solutions are sustained with high probability). Some of these bounds are also useful for the Pittsburgh approach of LCS, and the model proposed in this paper is inspired in the covering bound model of *XCS*.

The use of initialization operators that use training instance to create rules is not new in the LCS field. The covering operator performs this task in the *XCS* system [19]. This operator is activated when the system cannot match an input example and creates a rule that is a generalized version of the instance.

A similar operator is used in the *HIDER* system [1], which is an evolutionary iterative rule learning system. *HIDER* uses iteratively a GA to learn one rule at a time applying a separate-and-conquer policy. In this case the operator is used to create the initial population of the *GA*. These initial rules are also generalized versions of randomly sampled training instances.

We can also find examples of Pittsburgh approach systems that use similar operators, such as the *GIL* system [14]. This system proposes a mixed initialization stage, initializing randomly some of the individuals of the population and creating rules that are exact copies of randomly sampled training instances. This is the main difference from the two previously mentioned systems, where the created rules are *generalized* versions of the input instance, not *exact* copies. The *GIL* author states that the best initialization method is a mix of these two policies (random and instance-based). However, there is no information about how this mix is performed, nor a model about the initialization behaviour.

3. FRAMEWORK

The system used in this paper is called GAssist [3], and it is a Pittsburgh learning classifier system descendant of *GABIL* [10]. The system applies a near-standard *GA* that evolves individuals that represent complete problem solutions. An individual consists of a variable–length rule set. The knowledge representation for discrete attributes (rules with conjunctive normal form (CNF) predicates) and the semantically correct crossover operator is taken directly from *GABIL*. Next is a brief description of its main features.

Control of the individuals length: Dealing with variable-length individuals raises some important issues. One of the most important one is the control of the size of the evolving individuals because it can grow without control, something known as *Bloat effect* [18]. 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 an irreversible diversity loss) and (b) the number of rules of an individual never goes below a threshold.
- *Minimum description length-based fitness function.* The minimum description length (*MDL*) principle [17] is a metric applied in general to a theory (being a rule set in this paper) which balances the complexity and accuracy of the rule set. We have used an implementation of the *MDL* principle tailored to the Pittsburgh approach and the used knowledge representations [3].

Knowledge representations For nominal attributes the *GABIL* [10] representation has been used. For real-valued attributes we have used a representation called *adaptive discretization intervals* (*ADI*) rule representation [3]. This representation uses the semantics of the *GABIL* rules (conjunctive normal form predicates), but instead of the values of a nominal attribute, it uses non-static intervals formed by joining neighbouring low-level intervals provided by several predefined candidate discretization algorithms. These intervals can evolve through the learning process splitting or merging among them and potentially using several discretization algorithms at the same time.

Explicit and static default rule: Using the rules of an individual as an ordered set to perform the match process allows the creation of very compact rule sets by the use of default rules. We use an existing mechanism [5] to explicitly exploit this issue and determine automatically the class for the default rule.

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.

4. MODELLING THE MATCH PROBABILITY

The objective of this section is to model the probability that an individual has of matching a random input instance. Therefore, it is dependant of the knowledge representation. Thus, the first step is to describe briefly the used representation: the *GABIL* one.

In *GABIL* 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. In order to create the initial population, GABIL uses a probability of assigning value "1" to each position in the predicate.

Therefore, the probability that each initial rule has of matching a random input instance is defined in equation 1 where P_1 is the probability of value "1" and a is the number of attributes in the domain.

$$P(match) = (P_1)^a \tag{1}$$

We can extend 1 to a whole individual (consisting in a set of rules) with equation 2 where r is the initial number of rules per individual.

$$P(\text{match individual}) = 1-(1-P(match))^r = 1-(1-(P_1)^a)^r \tag{2}$$

Equation 2 would be applicable to the *GABIL* representation as it is defined. However, in the learning system studied in this paper we are also using an explicit default rule mechanism. This means that we have to use a P_1 probability low enough to make sure that the default rule is really used. Given the assumption of equal class distribution we can tune P_1 with the objective of having a match probability equal to the percentage of examples not covered by the default rule (where nc is the number of classes in the domain):

$$P(\text{match individual}) = 1 - \frac{1}{nc} \tag{3}$$

$$(1 - (P_1)^a)^r = \frac{1}{nc} \tag{4}$$

$$P_1 = \sqrt[a]{1 - \frac{1}{\sqrt[r]{nc}}} \tag{5}$$

Two constraints for the minimum and maximum allowed probability (0.1 and 0.95, respectively) are added. With 5 we have developed an automatic method to adjust the value for P_1. In order to check if the assumption made is valid enough we computed the experimental average match ratio of all the rules but the default one for the individuals in the initial population using the datasets used for the experimentation reported in section 6. Table 1 compares the experimental match ratio with the assumption of uniform class distribution, showing also the computed P_1. The results indicate that the average divergence is quite low (3.52%) showing that the assumption is quite reasonable.

Table 1: Comparison of the match ratio of the dynamic rules of the initial population with the uniform class distribution assumption

Name	Experimental	Assumption	P_1
bal	69.63%±1.01	66.67%	0.49
bpa	49.71%±1.72	50.00%	0.57
bre	51.17%±1.09	50.00%	0.69
cmc	70.51%±0.83	66.67%	0.73
col	52.36%±0.95	50.00%	0.86
cr-a	50.87%±0.99	50.00%	0.80
gls	86.55%±1.00	83.33%	0.77
h-c	55.48%±0.85	50.00%	0.78
h-h	55.46%±1.46	50.00%	0.78
h-s	55.22%±1.18	50.00%	0.78
hep	52.21%±1.00	50.00%	0.84
ion	56.27%±1.00	50.00%	0.91
irs	69.84%±1.39	66.67%	0.49
lab	49.96%±0.98	50.00%	0.81
lym	81.63%±0.88	75.00%	0.87
pim	52.22%±1.54	50.00%	0.66
prt	97.36%±0.33	95.24%	0.90
son	61.19%±0.83	50.00%	0.95
thy	67.89%±1.98	66.67%	0.56
vot	50.27%±1.14	50.00%	0.81
wbcd	51.68%±1.29	50.00%	0.69
wdbc	58.17%±1.52	50.00%	0.90
wine	67.71%±1.05	66.67%	0.80
wpbc	59.97%±1.31	50.00%	0.91
zoo	89.81%±0.75	85.71%	0.87
ave.	62.53%	58.91%	0.77

5. INSTANCE-BASED RULE INITIALIZATION

This smart initialization algorithm is defined in the code in figure 1. Basically, it samples without replacement an instance from the training set and creates a rule that is a generalization of the given instance (using P_1 to generalize the rule). It also takes into account the use of the explicit default rule, sampling only instances not belonging to the class of the default rule.

Two variants of the instance sampling mechanism have been tested. The first one assigns equal probability to all eligible instances, that is, not belonging to the default class of the individual being initialized. The second mechanism gives equal probability to all eligible classes. After randomly selecting the class, one of its instances is randomly sampled with uniform probability. The reason for testing this second mechanism is that the *GAssist* system has shown previously that has some difficulties in solving domains with high class

```
Procedure Rule initialization operator
Input : P_1, defaultClass, TrainingSet
Rule = create empty rule
Instance = sample without replacement an instance from
           TrainingSet not belonging to defaultClass
ForEach attribute in Instance
        instanceValue= Instance.value[attribute]
        ForEach value in attribute
                If value = instanceValue Then
                   Rule.attrs[attribute].val[value] = 1
                Else
                   If Rand[0,1] < P_1 Then
                      Rule.attrs[attribute].val[value] = 1
                   Else
                      Rule.attrs[attribute].val[value] = 0
                   EndIf
                EndIf
        EndForEach
EndForEach
Rule.class = Instance.class
Output : Rule
```

Figure 1: Smart rule initialization operator

unbalance [3]. It is clear that this mechanism will not completely solve the problem, but at least we can test if it has some influence over the performance of the system.

6. EXPERIMENTAL EVALUATION

6.1 Tests design

The objective of the experiments reported in this paper is to determine which is the most suitable (or at least more robust) global policy to automatically tune the initialization stage of *GAssist*. In order to achieve this objective we will combine the two kind of techniques studied in this paper:

- Policies to give value to P_1
  - The automatic policy based on the match probability model
  - Four global values of P_1: 0.25, 0.50, 0.75, and 0.90
- The rule initialization operator
  - Without smart initialization (labelled *NoSI*)
  - With smart initialization and instance-wise uniform sampling (labelled *IWSI*)
  - With smart initialization and class-wise uniform sampling (labelled *CWSI*)

The combination of these settings produce 15 configurations to test. 25 datasets have been used in this paper. These datasets represent a broad range of domains in respect to number of attributes, instances, type, etc. These problems were taken from the University of California at Irvine (UCI) repository [6], and their features are summarized in table 2.

The datasets will be partitioned using the standard stratified ten-fold cross-validation method. Also, three different sets of 10-cv folds and 5 random seed will be used. This means that the results for each dataset and configuration will be the average of 150 runs. Student t-tests will be used in order to analyze and compare the results of the tests, using a confidence interval of 95%. Also, the Bonferroni correction will be used for multiple pair-wise comparisons.

The comparison process will use the following methodology: First, for each of the three types of initialization operators tested we will use statistical tests to determine which of the 5 types of P_1 tuning policies is the most suitable one. After the best (or more robust) policy is identified for each initialization operator these optimal configurations will be compared among them to determine the global best policy. This global policy will be compared with some alternative learning systems.

Parameters of the system are described in table 3.

6.2 Results of the tests with NoSI

Table 4 shows the test accuracy for the tested P_1 tuning policies using the random initialization operator for all datasets. Table 5 shows global averages over all datasets of training and test accuracy, and also the results of the t-tests: the count of times that a method significantly outperforms other methods and the count of times that a method has been significantly outperformed.

From these results we can see a huge performance difference between the configurations using values 0.25 and 0.50

Table 2: Features of the datasets used in this paper. #Inst. = Number of Instances, #Attr. = Number of attributes, #Real = Number of real-valued attributes, #Nom. = Number of nominal attributes, #Cla. = Number of classes, Dev.cla. = Deviation of class distribution

Dataset Properties						
Code	#Inst.	#Attr.	#Real	#Nom.	#Cla.	Dev.cla.
bal	625	4	4	—	3	18.03%
bpa	345	6	6	—	2	7.97%
bre	286	9	—	9	2	20.28%
cmc	1473	9	2	7	3	8.26%
col	368	22	7	15	2	13.04%
cr-a	690	15	6	9	2	5.51%
gls	214	9	9	—	6	12.69%
h-c	303	13	6	7	2	4.46%
hep	155	19	6	13	2	29.35%
h-h	294	13	6	7	2	13.95%
h-s	270	13	13	—	2	5.56%
ion	351	34	34	—	2	14.10%
irs	150	4	4	—	3	—
lab	57	16	8	8	2	14.91%
lym	148	18	3	15	4	23.47%
pim	768	8	8	—	2	15.10%
prt	339	17	—	17	21	5.48%
son	208	60	60	—	2	3.37%
thy	215	5	5	—	3	25.78%
vot	435	16	—	16	2	11.38%
wbcd	699	9	9	—	2	15.52%
wdbc	569	30	30	—	2	12.74%
wine	178	13	13	—	3	5.28%
wpbc	198	33	33	—	2	26.26%
zoo	101	16	—	16	7	11.82%

Table 3: GAssist configuration for the tests reported in the paper

Parameter	Value
General parameters	
Crossover probability	0.6
Selection algorithm	Tournament
Tournament size	3
Population size	400
Individual-wise mutation probability	0.6
Initial #rules per individual	20
Rule Deletion operator	
Iteration of activation	5
Minimum number of rules	#active rules + 3
MDL-based fitness function	
Iteration of activation	25
Initial theory length ration	0.075
Weight relax factor	0.9
ADI rule representation	
Split and merge probability	0.05
Initial reinitialize probability	0.02
Final reinitialize probability	0
#bins of uniform-width discretizers	4,5,6,7,8,10,15,20,25
Maximum number of intervals	5

Table 4: Test accuracy of the tested P_1 tuning policies using *NoSI*. The best method for each dataset is marked in bold. A • symbol marks the methods being significantly outperformed by the best policy according to the t-tests

Dataset	25%	50%	75%	90%	Auto
bal	78.59±3.98•	78.97±4.08•	78.86±4.00	**79.33±3.96**	78.58±3.85
bpa	62.21±7.97	61.63±7.27	62.68±7.34	62.88±7.57	**63.13±7.74**
bre	68.19±9.26•	68.46±9.00	69.48±8.28	**70.62±7.56**	69.40±7.56
cmc	54.89±4.03	55.03±4.15	**55.10±4.01**	54.40±4.02	55.02±4.10
col	63.07±1.52•	70.49±13.47•	**93.59±3.94**	93.48±4.57	93.46±4.40
cr-a	55.71±2.43•	**85.39±3.92**	85.10±4.07	85.29±3.69	85.05±3.81
gls	62.82±12.17•	67.52±10.04	67.41±9.97	**68.35±8.72**	67.92±9.48
h-c	55.38±4.76•	**80.45±5.99**	79.40±5.68	80.35±6.33	80.34±6.08
h-h	64.57±4.94•	**96.14±3.22**	95.71±3.18	95.95±3.51	95.83±3.35
h-s	57.11±6.71•	80.47±7.49	80.52±7.83	**80.59±7.35**	80.54±7.18
hep	79.38±2.15•	82.11±5.85•	89.16±8.15	89.70±7.98	**89.82±8.20**
ion	64.12±1.56•	64.12±1.56•	92.72±4.85	**92.91±4.58**	92.56±4.98
irs	95.16±5.70	95.07±5.88	**95.33±5.70**	94.89±5.73	95.07±5.73
lab	64.25±4.66•	95.09±10.96•	97.51±6.29	**98.17±5.36**	97.36±6.26
lym	54.86±3.03•	69.67±14.79•	**80.47±10.35**	80.23±10.24	80.44±10.76
pim	74.33±4.52	74.36±4.45	74.28±4.54	74.77±5.14	**74.90±4.71**
prt	28.94±5.48•	43.13±9.81•	47.43±6.70	**48.24±7.61**	47.93±7.28
son	53.43±2.25•	53.43±2.25•	54.73±6.55•	75.83±8.88	**76.43±9.94**
thy	91.88±5.99	91.72±5.86	**92.15±5.50**	92.10±5.91	92.09±5.91
vot	61.39±1.29•	96.74±3.57	97.09±3.22	**97.30±2.65**	96.97±2.90
wbcd	96.06±2.42	96.19±2.45	96.00±2.38	96.00±2.26	**96.20±2.28**
wdbc	62.74±0.62•	62.74±0.62•	**94.30±2.67**	94.24±2.91	94.14±2.99
wine	41.02±7.51•	93.05±5.90	93.35±5.66	92.64±5.63	**93.69±5.17**
wpbc	76.44±3.68	76.44±3.68	**76.51±7.85**	75.89±8.13	74.85±8.60
zoo	41.10±5.04•	78.40±20.52•	91.29±8.44	91.28±8.83	**92.56±8.38**

Table 5: Averages of training and test accuracy and results of the t-tests (#times outperforming/#times outperformed) for the tested P_1 tuning policies using *NoSI*

P_1 policy	Training acc.	Test acc.	T-tests
0.25	67.65±17.95	64.31±15.83	0/61
0.50	82.77±14.80	76.67±14.74	13/25
0.75	89.34±13.29	81.61±14.59	22/2
0.90	90.78±11.50	82.62±13.48	28/0
Auto	91.05±11.38	82.57±13.53	25/0

Table 7: Averages of training and test accuracy and results of the t-tests (#times outperforming/#times outperformed) for the tested P_1 tuning policies using *IWSI*

P_1 policy	Training acc.	Test acc.	T-tests
0.25	88.83±13.03	80.76±14.43	0/18
0.50	90.20±11.70	81.77±14.05	3/9
0.75	91.05±11.29	82.50±13.45	8/0
0.90	90.84±11.47	82.59±13.58	8/0
Auto	91.02±11.41	82.54±13.52	8/0

for P_1 and the other configurations. Looking at the full results in table 4 we can see how some configurations are unable to learn at all in certain datasets. An extreme example of this problem is the *son* dataset which needs at least a value of 0.90 for P_1. The only two configurations that are able to learn properly for all the tested datasets are 0.75 and the automatic one.

The automatic policy manages to obtain the highest training accuracy, which indicates that it is able to produce a set of initial individuals that allow the system to learn but that are not over-general, showing a good equilibrium between generality and specificity. However, this fact does not produce an advantage in the test stage, where this configuration achieves very similar performance to the 0.75 and 0.90 policies. The combination of t-tests and average accuracy results indicate that the best policy is 0.90.

6.3 Results of the tests with IWSI

Tables 6 and 7 summarize the results of testing the smart initialization operator with instance-wise sampling. The results show how the smart initialization operator has a positive effect on all the configurations that were unable to learn in the previous tests with the random initialization operator. We can observe that two configurations (0.75 and Auto) have higher training accuracy than the configuration with top test accuracy (0.90). The smart initialization operator amplifies the effect observed in the Auto configuration of the previous tests. The combination of t-tests and average accuracy results indicate that the best policy is 0.90.

6.4 Results of the tests with CWSI

Tables 8 and 9 summarize the results of testing the smart initialization operator with class-wise sampling. The results show how all configurations increase or maintain the test accuracy in respect to the previously reported results about the smart initialization operator using instance-wise sampling. This time the only configuration that has never been significantly outperformed is 0.75, and unlike the two previous groups of tests, this configuration almost achieves the top training accuracy.

6.5 Comparison of the initialization operators

This subsection summarizes the results of the previous three subsections by comparing the best P_1 policies for the three studied initialization operators. Table 10 contains this comparison. We can see how all methods achieve similar test performance, being IWSI and CWSI slightly more robust than NoSI.

Table 6: Test accuracy of the tested P_1 tuning policies using *IWSI*. The best method for each dataset is marked in bold. A • symbol marks the methods being significantly outperformed by the best policy according to the t-tests

Dataset	25%	50%	75%	90%	Auto
bal	78.23±3.91	78.70±3.99	79.01±4.03	78.92±3.57	**79.05±3.65**
bpa	62.63±8.32	62.09±8.60	62.93±7.78	62.47±8.59	**63.56±7.04**
bre	64.59±8.62•	66.71±9.50•	**70.65±7.80**	70.26±7.70	69.30±8.77
cmc	54.82±4.22	54.87±4.00	54.89±4.06	54.43±3.91	**54.90±4.13**
col	91.65±5.07•	92.11±4.71	92.53±4.60	**93.12±4.41**	92.70±4.70
cr-a	84.96±3.91	85.19±3.81	85.15±3.90	85.18±4.00	**85.41±3.94**
gls	66.54±9.73	**67.58±9.31**	66.74±9.37	66.80±10.03	67.45±9.10
h-c	80.38±6.41	80.78±6.30	80.70±5.70	**80.94±5.60**	80.79±5.95
h-h	95.93±3.46	95.83±3.14	95.95±3.36	95.97±3.51	**96.14±3.12**
h-s	80.40±7.57	**80.64±7.09**	80.15±6.97	79.70±7.66	79.75±7.23
hep	89.42±7.56	89.43±7.31	89.96±7.57	89.70±8.38	**90.06±7.80**
ion	85.86±8.80•	91.77±5.39•	**92.71±4.88**	92.63±5.08	92.25±4.84
irs	94.89±5.83	**95.16±5.91**	94.58±5.88	95.07±5.78	95.07±5.52
lab	96.37±7.06	96.94±6.86	97.75±5.79	97.77±6.63	**97.82±5.79**
lym	76.93±10.46•	79.27±11.07•	80.15±11.11	**82.47±9.25**	81.06±11.31
pim	74.33±4.91	74.64±4.45	74.62±4.79	**74.72±4.77**	74.49±5.09
prt	**49.24±7.09**	48.46±7.13	48.84±6.96	48.22±6.96	48.21±7.00
son	53.83±3.78•	62.98±11.31•	74.30±9.28	76.32±9.29	**76.68±9.09**
thy	91.98±5.41•	91.57±5.63•	91.91±5.40	92.06±5.82	**92.10±5.37**
vot	96.42±3.69	96.50±3.40	97.08±3.35	96.97±3.08	**97.28±3.01**
wbcd	95.95±2.47	96.05±2.53	95.92±2.46	95.93±2.40	**96.16±2.38**
wdbc	94.27±3.15	93.74±3.05	94.21±3.00	**94.36±2.80**	94.27±2.88
wine	92.48±5.88	**93.89±5.18**	92.93±5.00	93.18±5.07	93.35±5.66
wpbc	76.02±8.08	75.87±8.04	**76.04±9.30**	75.38±8.17	74.10±8.51
zoo	91.00±8.59•	**93.54±7.37**	92.75±7.29	92.17±8.47	91.51±8.18

Table 8: Test accuracy of the tested P_1 tuning policies using *CWSI*. The best method for each dataset is marked in bold. A • symbol marks the methods being significantly outperformed by the best policy according to the t-tests

Dataset	25%	50%	75%	90%	Auto
bal	78.45±3.51	78.65±3.61	**79.33±3.92**	79.06±4.25	78.76±4.29
bpa	62.89±7.45	62.09±8.13	**63.14±8.62**	62.54±7.40	62.01±7.39
bre	66.09±9.52•	67.73±8.21•	69.73±7.60	**71.37±8.03**	69.51±8.57•
cmc	55.02±4.25	**55.35±4.08**	54.97±3.78	54.58±3.85•	54.93±4.05
col	91.50±5.30•	92.06±5.02	92.62±4.81	92.85±5.03	**93.30±4.81**
cr-a	85.15±3.88	85.04±4.04	85.29±3.94	85.29±3.94	**85.38±3.84**
gls	68.08±8.66	67.38±10.09	68.13±9.21	66.61±9.79	**68.47±9.02**
h-c	79.99±6.58	80.79±6.44	80.25±5.91	80.64±5.78	**81.42±5.87**
h-h	96.20±3.32	95.71±3.62	95.54±3.67	**96.16±3.20**	95.67±3.31
h-s	79.98±7.76	**81.09±7.48**	80.15±7.47	79.78±6.98	80.54±7.70
hep	89.82±6.90	**90.55±7.67**	90.50±6.87	90.54±7.98	89.10±7.92
ion	84.11±9.97•	91.99±5.39	**92.66±4.95**	92.65±4.76	92.44±5.11
irs	95.29±5.63	95.02±5.77	**95.42±5.77**	95.07±5.78	95.11±5.48
lab	96.18±7.27	97.76±5.95	**97.96±5.61**	97.72±5.88	97.49±6.12
lym	78.38±11.40•	77.98±11.16•	**81.23±9.98**	80.87±10.33	81.05±9.85
pim	74.34±4.61	74.24±4.75	**74.77±4.40**	74.31±4.74	74.54±4.92
prt	47.93±6.93	**48.80±7.56**	48.25±7.47	47.51±6.81	48.07±7.18
son	54.45±5.09•	64.41±11.89•	76.50±9.84	76.44±8.87	**76.71±9.44**
thy	91.96±5.10	92.04±5.21	91.56±5.87	**92.35±5.85**	91.46±5.49
vot	96.12±3.52•	96.38±3.40•	96.82±3.52	**97.23±2.75**	97.16±2.81
wbcd	96.03±2.40	95.91±2.38	96.00±2.41	95.99±2.33	**96.07±2.42**
wdbc	94.00±2.69	94.08±2.96	**94.53±3.11**	94.18±2.97	94.40±2.76
wine	92.75±5.22	93.30±5.63	93.12±5.20	**93.57±5.56**	92.56±5.88
wpbc	75.57±7.30	**76.67±8.33**	76.60±7.91	75.30±9.04	75.22±8.68
zoo	92.66±8.20	**93.59±7.09**	93.05±7.24	92.48±7.24	92.04±8.46

6.6 Comparing the best P_1 policy with some alternative learning systems

In this subsection we compare the best *GAssist* configuration (CWSI) to some alternative learning systems that represent several type of knowledge representations and learning paradigms:

- **C4.5** [16], the well known decision tree induction learning system
- **IBk** [2], a nearest neighbour classifier using k=3
- **NaiveBayes** [15], a Bayesian network approach using a non-parametric kernel density estimator for real-valued attributes
- **LIBSVM** [9], a support vector machine

Table 9: Averages of training and test accuracy and results of the t-tests (#times outperforming/#times outperformed) for the tested P_1 tuning policies using *CWSI*

P_1 policy	Training acc.	Test acc.	T-tests
0.25	88.74±13.00	80.92±14.39	0/17
0.50	90.29±11.57	81.94±13.93	3/9
0.75	91.05±11.34	82.72±13.44	7/0
0.90	90.82±11.47	82.60±13.67	9/1
Auto	91.08±11.42	82.54±13.48	9/1

Table 10: Averages of training and test accuracy and results of the t-tests (#times outperforming/#times outperformed) for the best P_1 tuning policies for each studied initialization operator

P_1 policy	Training acc.	Test acc.	T-tests
NoSI	90.78±11.50	82.62±13.48	0/2
IWSI	90.84±11.47	82.59±13.58	1/0
CWSI	91.05±11.34	82.72±13.44	1/0

With the exception of *LIBSVM*, we have used the *WEKA* [20] implementation of these algorithms, using the default parameters. Table 11 summarizes the results of this comparison. We can see how *GAssist* is the top performing method and also the learning system least times being significantly outperformed (almost tied with *LIBSVM*. As expected given the *selective superiority problem* [7], there is no learning system completely superior to the other ones. Nevertheless the results show how *GAssist* has competitive performance compared to the other systems.

As a reference for the results reported in this paper, there is a recent comparison of a basic *GAssist* version (using *NoSI*) against *XCS* in the literature [4]. Results of *XCS* are not included in this paper because different partitions of the datasets where used therefore the statistical tests used would not be directly applicable.

7. DISCUSSION

The reported results show some general trends. First of all, *GAssist* needs very general initial individuals (very high P_1 probability) in order to learn properly. This requirement can be slightly relaxed when the smart initialization operator is used, specially when applying the class-wise sampling. Also, observing the results for each of the three tested initialization operators we can see how the configuration achieving top training accuracy is not the best one in the test stage.

These two facts indicate that the goal of the initialization policies should not be to generate the most accurate initial individuals, but to create a set of individuals that by means of the current crossover and mutation operators can lead to well generalized final solutions. By this we mean that the system right now is not capable of merging accurate but low-supported rules to create more general and equally accurate rules. Nevertheless, this is not a critical drawback because the results show how *GAssist* has competent performance. It only reflects the initialization requirements of the system.

Table 11: Averages of training and test accuracy and results of the t-tests (#times outperforming/#times outperformed) for the comparison of GAssist with some alternative learning systems

P_1 policy	Training acc.	Test acc.	T-tests
GAssist	91.05±11.34	82.72±13.44	44/28
C4.5	90.47±9.29	79.45±13.06	19/59
IBk	88.84±9.39	81.34±14.09	34/42
NaiveBayes	84.30±12.87	81.23±14.44	43/33
LIBSVM	84.44±14.37	80.99±15.38	61/29

It is necessary to analyze if it is worth using the smart initialization operator. The results show how it is only significantly better than the original random initialization in one of the 25 tested datasets. This operator has to be combined with a high P_1 probability in order to achieve good test accuracy, and this means that the influence of the operator is smaller because the generated rules are not that much different from a pure randomly initialized rule.

Nevertheless, we think that this is only a first step. If this operator can be combined, for instance, with a fitness function biased towards individuals that cover all classes in the dataset, the performance of *GAssist* in domains with high class unbalance can be improved. Also, if the system could be extended with smart recombination operators that do not require such high P_1 probability, the initialization policies could be tuned to more aggressive (less over-general) settings in order to boost the performance of the system in both training and test accuracy.

8. CONCLUSIONS AND FURTHER WORK

This paper has studied the initialization stage of the Pittsburgh approach of learning classifier systems. The paper started by proposing a model for the match probability of a random rule set (individual) for the *GABIL* knowledge representation. The model was applied to automatically tune P_1, the initialization probability used in *GABIL*, which led to the second part and major objective of the paper: determine empirically if there is a global initialization policy that is suitable for a broad range of datasets.

The experimentation contained two kinds of tested techniques: (1) P_1 tuning policies, comparing the automatic policy proposed with some global values for P_1 and (2) initialization operators, comparing the purely random rule initialization operator against an smart one generating rules that are a generalized version of a randomly sampled training instance. Two types of instance sampling methods were tested: a pure sampling without replacement of all instance in the training set and a sampling process where all classes in the dataset have equal probability of being used.

The combination of these two kinds of policies led to 15 configurations being tested. We tested separately the P_1 tuning policies for each initialization operator in order to determine the most suitable policy for each of them. Later, when comparing the initialization operators we used only these best policies. The objective was to determine if there is some general initialization policy with competent performance, in order to simplify the tuning process needed to adjust the system for each dataset. This objective has been completely achieved, and by more than a single configuration. The results also indicate the kind of initial individuals that are required in order to guarantee that the system creates well-generalized solutions.

The main priorities of the further work should be to modify the system to exploit more successfully the techniques studied in this paper by means of smart recombination operators and class-wise fitness functions. It would also be worth to use the match probability model from others points

of view. For simplicity reasons, we have used only a single value for the initial number of rules per individual. It would be worth to study the influence of this variable in the performance of the system. Moreover, the model could be used to create a population sizing formula. A model relating the generality degree of the individuals with the success of the crossover operator would also be useful to complement the existing match probability model and refine the P_1 tuning formula.

9. ACKNOWLEDGEMENTS

The author would like to thank Martin Butz for his useful comments, and acknowledge the support provided by the Spanish Research Agency (CICYT) under grant numbers TIC 2002-04160-C02-02 and TIC 2002-04036-C05-03, the support provided by the Department of Universities, Research and Information Society (DURSI) of the Autonomous Government of Catalonia under grants 2002-SGR-00155 and 2001-FI-00514.
Also, this work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, and by the Technology Research, Education, and Commercialization Center (TRECC), at University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research under grant N00014-01-1-0175. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the Technology Research, Education, and Commercialization Center, the Office of Naval Research, or the U.S. Government.

10. REFERENCES

[1] J. Aguilar-Ruiz, J. Riquelme, and M. Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 33(2):324–331, April 2003.

[2] D. W. Aha, D. F. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[3] J. Bacardit. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain, 2004.

[4] J. Bacardit and M. V. Butz. Data mining in learning classifier systems: Comparing xcs with gassist. In *Proceedings of the 7th International Workshop on Learning Classifier Systems*. (in press), LNAI, Springer-Verlag, 2004.

[5] J. Bacardit, D. E. Goldberg, and M. V. Butz. Improving the performance of a pittsburgh learning classifier system using a default rule. In *Proceedings of the 7th International Workshop on Learning Classifier Systems*. (in press), LNAI, Springer-Verlag, 2004.

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

[7] C. Brodley. Addressing the selective superiority problem: Automatic algorithm /model class selection. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 17–24. Morgan Kaufmann Publishers, 1993.

[8] M. V. Butz. *Rue-based Evolutionary Online Learning Systems: Learning Bounds, Classification and Prediction*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[9] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*. Department of Computer Science and Information Engineering, National Taiwan University, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] K. A. DeJong, W. M. Spears, and D. F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13(2/3):161–188, 1993.

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

[12] D. E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.

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

[14] C. Janikow. *Indictive Learning of Decision Rules in Attribute-Based Examples: a Knowledge-Intensive Genetic Algorithm Approach*. PhD thesis, University of North Carolina, 1991.

[15] G. H. John and P. Langley. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann Publishers, San Mateo, 1995.

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

[17] J. Rissanen. Modeling by shortest data description. *Automatica*, vol. 14:465–471, 1978.

[18] T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evolutionary Computation*, 6(4):293–309, Winter 1998.

[19] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[20] I. H. Witten and E. Frank. *Data Mining: practical machine learning tools and techniques with java implementations*. Morgan Kaufmann, 2000.