Smart Crossover Operator with Multiple Parents for a Pittsburgh Learning Classifier System

Jaume Bacardit
ASAP research group, School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK
jqb@cs.nott.ac.uk

Natalio Krasnogor
ASAP research group, School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, NG8 1BB, UK
nxk@cs.nott.ac.uk

ABSTRACT

This paper proposes a new smart crossover operator for a Pittsburgh Learning Classifier System. This operator, unlike other recent LCS approaches of smart recombination, does not learn the structure of the domain, but it merges the rules of N parents ($N \geq 2$) to generate a new offspring. This merge process uses an heuristic that selects the minimum subset of candidate rules that obtains maximum training accuracy. Moreover the operator also includes a rule pruning scheme to avoid the inclusion of over-specific rules, and to guarantee as much as possible the robust behaviour of the LCS. This operator takes advantage from the fact that each individual in a Pittsburgh LCS is a complete solution, and the system has a global view of the solution space that the proposed rule selection algorithm exploits. We have empirically evaluated this operator using a recent LCS called GAssist. First with the standard LCS benchmark, the 11 bits multiplexer, and later using 25 standard real datasets. The results of the experiments over these datasets indicate that the new operator manages to increase the accuracy of the system over the classical crossover in 16 of the 25 datasets, and never having a significantly worse performance than the classic operator.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept Learning, Induction*

General Terms

Algorithms,Experimentation,Performance

Keywords

Evolutionary Algorithms, Learning Classifier Systems, Rule Induction, Smart Crossover operator

1 Introduction

For several years, a strong line of research in the evolutionary computation field is the proposal of smart recombination operators, with many different approaches, such as *estimation of distribution algorithms* [10] or *competent genetic algorithms* [6], among others. In the learning classifier systems (LCS) field, this line of research has started much recently [4], when the well-known XCS system [15] was extended with a crossover operator based on two types of competent genetic algorithms. These approaches are mainly based on learning the relationships among the variables of the problem.

This paper presents an smart crossover operator (from now on, **SX**) applied to the other main family of LCS, the Pittsburgh approach. Unlike most of the previous approaches, we will not apply any learning or statistical technique to determine the structure of the problem: SX will use all the rules of N parents ($N \geq 2$) and will heuristically select the minimum subset of rules that obtains maximum accuracy over the training set, also deciding the order of the rules. In our system it is easy to determine, at some degree, the fitness contribution of each part of the global solution (a rule). Therefore, we will recombine in a smart way the already existing rules in the population. This approach is quite different to the smart recombination technique used in *XCS*, that was applied at a rule-level. In this sense, we exploit the main characteristic of the Pittsburgh LCS: Each individual is a complete solution, therefore SX can have a global view of the solution space.

We focus on a recent system belonging to the Pittsburgh approach of LCS, called GAssist [1]. Through the paper, we describe the basic approach of SX, we test this basic version with some theoretical and empirical challenges and we propose some modifications to overcome the detected weak points. The final version of SX is tested on a set of standard 25 datasets that represent a broad range of scenarios and it is compared to the classic crossover operator (from now on, **CX**) from several points of view. SX, although having more computational cost, improves the performance of GAssist in 16 of the 25 tested datasets (outperforming significantly CX in four of them) and it never significantly degrades the performance of the system.

The rest of the paper is structured as follows: First, section 2 will describe some related work. Next, section 3 will contain the main characteristics of GAssist, the Pittsburgh LCS used in this paper. Section 4 will explain the development of the SX presented in this paper. After the definition of the new operator, section 5 will describe the experiments done to evaluate its performance and analyze the results of these experiments. Section 6 will describe the conclusions and further work of the paper.

2 Related work

The use of informed/wise/smart recombination operators has been quite widespread in the EC community for some years. As stated above two of such approaches are *estimation of distribution algorithms (EDA)* [10] or *competent genetic algorithms* [6]. Usually these paradigms involve applying machine learning or statistic techniques to learn the structure of the problem being solved and allow the system to explore better the search space by creating informed exploration operators.

However, until two years ago this line of research had rarely been used in the LCS field. One of such approaches [4] was applied to the XCS system [15], that was extended with a crossover operator based on two types of competent genetic algorithms: the Extended Compact Genetic Algorithm [8] and the Bayesian Optimization Algorithm [12]. These two methods derive global structural information from the best rules in the population, and later XCS uses this information to generate offspring in a smart way.

The Compact Classifier System (CCS) [11] is a recent approach of using EDAs within the framework of a Pittsburgh LCS. The selected EDA was the Compact Genetic Algorithm [9]. CGA is run iteratively to generate different rules. Different perturbations of the initial solution of CGA are needed to generate different rules, and the individuals in CCS store a set of such perturbations. The objective of CCS is to determine the minimum set of rules that creates a maximally general solution. Other early attempts at using EDAs in a Pittsburgh LCS are also discussed in [11].

Nevertheless, the approach in this paper is probably more close to what could be called local search or heuristic crossover instead of really smart crossover. In this scope, there is a quite early LCS work, the SAMUEL system [7], that has a similar operator to the one presented here. That system was applied to multi-step domains, and their operator generated an offspring containing high-payoff rules that fired in sequence. However, their operator used rules only from two parents. Also, it was applied to unordered rules, while our approach is specifically designed for ordered rules.

3 The GAssist Learning Classifier System

GAssist [1] is a Pittsburgh Learning Classifier System descendant of GABIL [5]. The system applies a near-standard generational GA that evolves individuals that represent complete problem solutions. An individual consists of an ordered, variable–length rule set.

A special fitness function based on the Minimum Description Length (MDL) principle [13] is used. The MDL principle is a metric applied in general to a theory (being a rule set here) which balances the complexity and accuracy of the rule set The details and rationale of this fitness formula are explained in [1]. The system also uses a windowing scheme called ILAS (incremental learning with alternating strata) to reduce the run-time of the system, but also to introduce extra generalization pressure that improves the generalization capacity of the system. This mechanism divides the training set into several non-overlapped subsets and chooses a different subset at each GA iteration for the fitness computations of the individuals.

We have used the GABIL [5] rule-based knowledge representation for nominal attributes and the adaptive discretization intervals (ADI) rule representation [1] for real-valued ones. To initialize each rule, the system chooses a training example and creates a rule that covers this example, using the mechanism proposed at [2]. The representation is extended with an explicit default rule mechanism: 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 [1] to explicitly exploit this effect.

4 The smart crossover operator

This section is divided in three parts. The first one describes the development of the operator itself. The second one details how SX has been integrated into the framework of GAssist. Finally, the third part illustrates experimentally the strong and weak points of the operator, and proposes possible ways to overcome the current weaknesses.

4.1 Definition of the operator

SX has three main stages, shown by figure 1:

1. Evaluation of the candidate rules

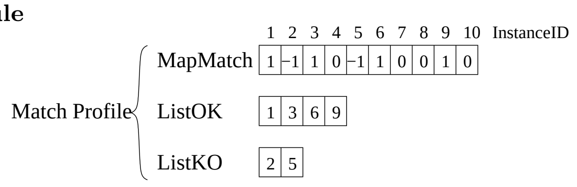
2. Selection of the offspring rule-set

3. Generation of the final individual

The rest of the subsection will focus on the first and second stage, as the third one is just the trivial generation of the offspring, once we have selected which rules will be joined together to create it. Depending on the employed knowledge representation it might be necessary to recompute some statistics about the new rule set.

In the first stage, the candidate rules from all parents are evaluated with all the training set. This process of evaluation, which is the most costly part of the algorithm, is performed only once. From each evaluation a data structure called "match profile" is be generated. Each match profile (shown in figure 2) contains a map of the training set indicating which examples are correctly or incorrectly classified by the rule and which of them are not matched. The process of selecting the new rule-set for the offspring uses these match profiles instead of re-evaluating the rules.

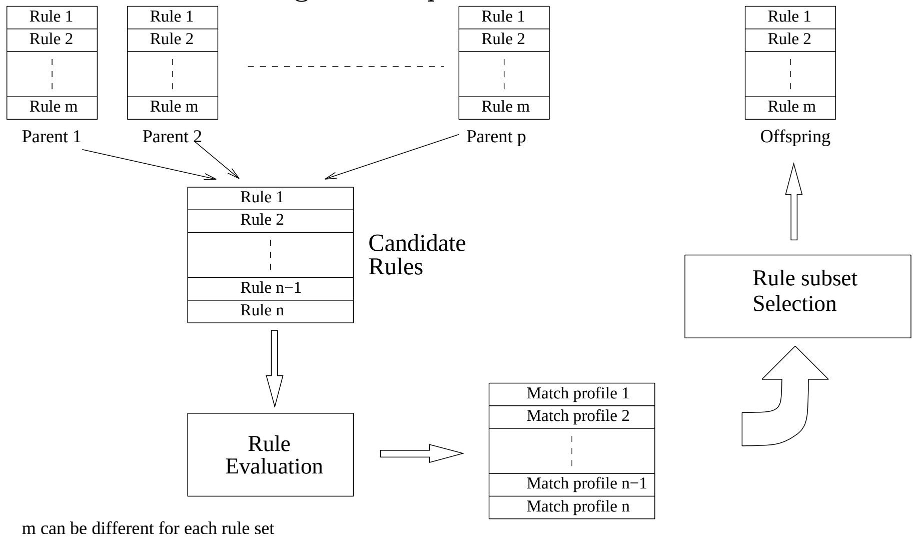
Figure 2: Representation of the match profile of a rule



After generating the profiles of the candidate rules, the central part of the algorithm starts. The procedure is quite simple: The process starts with an empty rule set, where all instances are matched by the default rule. Next, for each candidate rule, we look for the position in the rule set where this rule can help to maximize the training accuracy of the whole rule set. If the obtained accuracy is higher than the previous accuracy of the rule set without the new rule, it is inserted in that position. If not, the rule is discarded. In order to avoid any possible bias, we shuffle the candidate rules before trying to insert each of them into the new rule set. We will illustrate later with some examples why is this shuffling process necessary.

In order to reduce the run-time of the algorithm, the computation of the accuracy of each tested rule in the different

Figure 1: Representation of SX



positions of the rule set will be computed incrementally. To support this incremental computation we will use two data structures. The first one indicates, for each instance in the training set, if the example is correctly or incorrectly classified in the current rule set. The second structure indicates which rule matches each example. These data structures are initialized in a way that all examples are matched by the default rule, some of them correctly, some of them not. These two structures plus a list of the already selected rules and the current training accuracy of these rules are grouped in a data structure called "ruleOrder".

After trying to insert all candidate rules in the rule set, the selected rule subset is created by removing the inserted rules that have been subsumed during the process and do not contribute to increase the training accuracy anymore. Figures 3 and 4 contains the pseudo-code of the whole rule selection algorithm.

Please, note that the position where each candidate rule is inserted is not the position where this rule obtains maximum accuracy, but the position that maximizes the global training accuracy of the rule set. In this way we exploit as much as possible the advantages of having individuals that are complete solutions to the domain. Also note that the algorithm only inserts rules if they contribute to the rule set accuracy. The aim of this policy is two-fold: (1) avoiding the insertion of rules with high error rate and (2) also avoiding the insertion of rules that can be subsumed by other candidate rules. Let us illustrate this with two examples:

- We have two rules A and B. Rule B covers a subset of the examples covered by A. For simplicity we suppose that neither of the rules cover any negative example
- We try to insert first A and then B
  - A is inserted in some position of the rule set, following the specified algorithm
  - As B just covers a subset of A, there is no position in the rule set where B can help increase the training accuracy, because A is already in the rule set. Therefore, B is discarded
- We try to insert first B and then A
  - B is inserted in some position of the rule set, following the specified algorithm
  - A is inserted before B as the algorithm will always insert the rule in the top-most position that maximizes accuracy
  - In the cleaning-up stage of the algorithm, B will be removed because it does not match anymore any example

This example illustrates how this algorithm always tries to select the most general but accurate rules to generate the new rule set for the offspring. However, it has some limitations, illustrated by the second theoretical example:

- We have three rules, A, B and C. Rules B and C cover together the same examples as A
- Order of insertion: A, B, C or A, C, B
  - A is inserted in some position of the rule set
  - Neither B or C are inserted as they do not make the rule-set increase its training accuracy
- We try to insert first either B or C, then A and then C or B
  - B or C is inserted in some position of the rule set
  - A is inserted before B or C as the algorithm will always insert the rule in the top-most position that maximizes accuracy
  - C or B is not inserted
  - In the cleaning-up stage of the algorithm, B or C will be removed because it does not match anymore any example.
- We try insert first B and C (in any order) and then A
  - B and C are inserted in some position of the rule set
  - As A covers the same examples as the union of B and C, we cannot find any position where A increases the fitness of the rule set. Therefore, A is never inserted

Figure 3: Pseudo-code of the Rule Selection algorithm - first part

```
Procedure RuleSelectionAlgorithm
Input : $MatchProfiles$, $Examples$, $DefaultClass$
Shuffle $MatchProfiles$
$RuleOrder$ = InitializeRuleOrder($Examples$,$DefaultClass$)
ForEach $profile$ in $MatchProfiles$
        ($newAccuracy$,$position$) = FindBestPosition($RuleOrder$,$profile$)
        If $newAccuracy > RuleOrder.accuracy$
            $RuleOrder$=InsertRule($RuleOrder$,$profile$,$position$)
        EndIf
EndFor
Eliminate from $RuleOrder.selectedRules$ those rules that do not match any example
Output : $RuleOrder.selectedRules$,$RuleOrder.accuracy$

Procedure InitializeRuleOrder
Input : $Examples$,$DefaultClass$
$RuleOrder.selectedRules = \emptyset$
$RuleOrder.accuracy$ = accuracy of classifying all examples using $DefaultClass$
$RuleOrder.exampleClassifiedOK$ = initialize map of size $Examples.size$
$RuleOrder.exampleMatchedByRule$ = initialize map of size $Examples.size$
ForEach $example$ in $Examples$
        $RuleOrder.exampleMatchedByRule[example] = 0$
        If $example.class = DefaultClass$
            $RuleOrder.exampleClassifiedOK[example] = true$
        Else
            $RuleOrder.exampleClassifiedOK[example] = false$
        EndIf
EndFor
Output : $RuleOrder$

Procedure FindBestPosition
Input : $RuleOrder$,$profile$
$bestPosition = 0$
$bestAccuracy$ = AccuracyAtPosition($RuleOrder$,$profile$,0)
For $position$=1 to $RuleOrder.selectedRules.length - 1$
    $accuracy$ = AccuracyAtPosition($RuleOrder$,$profile$,$position$)
    If $accuracy > bestAccuracy$
        $bestAccuracy = accuracy$
        $bestPosition = position$
    EndIf
EndFor
Output : $bestAccuracy$,$bestPosition$
```

In the second example there is one case where we end up with a sub-optimal rule-set. How can we guarantee that we do not try to insert A as the last rule? In this paper we use the following heuristic: We repeat several times the process of creating the offspring rule set with different initial orderings, and we will select the rule set that have maximum accuracy. If we have two candidate rule sets with the same accuracy, we select the one with smaller number of rules. The pseudo-code for the whole operator including this improvement is represented in figure 5.

4.2 Integration of the operator in GAssist crossover stage

Analyzing the detailed pseudo-codes presented it is possible to see that SX only recombines already existing rules, and it does not generates new ones (as the standard crossover does). Therefore, we need to mix both kinds of crossovers. This section describes the integration of the new SX into GAssist and its combination with the traditional crossover. The integration of both crossovers occurs through a random control variable (P_{SmartX}) that stochastically selects which one to use. To prevent SX from choose the same parent twice in the same offspring generation process, we sample without replacement the parents.

4.3 Illustration of the smart crossover performance and robustness improvement proposal

This subsection illustrates the performance of the SX with some plots and results tables. First we show how this operator improves the performance of GAssist in a few datasets. Later we show its main weakness, namely overlearning, and we propose a heuristic that alleviates this weakness.

The first dataset that we test is the most standard benchmark in the LCS community: the 11 bit multiplexer. This is an easy test, as CX can solve successfully this dataset and finds the optimal solution always. However, the new operator helps GAssist converge towards the optimal solution using less iterations. Figure 6 illustrates the GAssist performance testing separately the three parameters of SX: number of parents, number of repetitions of the rule selection process and probability of smart crossover. Each plot is the average of 10 runs.

The next two datasets are *bal* and *zoo*, detailed in next section. These two datasets are quite small. However, GAssist had showed in the past some difficulties in learning them [1]. Table 1 reports the accuracy for both datasets testing all combination of these values for the parameters for SX:

- Number of parents: 2, 5 and 10
- Number of repetitions of the rule selection process: 1, 2 and 5
- Probability of Smart Crossover: 0.05 and 0.10

Figure 4: Pseudo-code of the Rule Selection algorithm - second part

```
Procedure AccuracyAtPosition
Input : $RuleOrder$,$profile$,$position$
$newAccuracy = RuleOrder.accuracy$
ForEach $example$ in $Profile.listOK$
        If $RuleOrder.exampleClassifiedOK[example] = false$
          and $RuleOrder.exampleMatchedByRule[example] \geq position$
          Increase $newAccuracy$ by one example
        EndIf
EndFor
ForEach $example$ in $Profile.listKO$
        If $RuleOrder.exampleClassifiedOK[example] = true$
          and $RuleOrder.exampleMatchedByRule[example] \geq position$
          Decrease $newAccuracy$ by one example
        EndIf
EndFor
Output : $newAccuracy$

Procedure InsertRule
Input : $RuleOrder$,$profile$,$position$
Insert rule associated to $profile$ in $RuleOrder.selectedRules$ at position $position$
ForEach $example$ in $RuleOrder.exampleMatchedByRule$
        If $RuleOrder.exampleMatchedByRule[example] >= position$
          $RuleOrder.exampleMatchedByRule[example] ++$
          If $profile.MatchMap[example] = 1$
            $RuleOrder.exampleMatchedByRule[example] = position$
            If $RuleOrder.exampleClassifiedOK[example] = false$
              Increase $RuleOrder.accuracy$ by one example
            EndIf
          EndIf
          If $profile.MatchMap[example] = -1$
            $RuleOrder.exampleMatchedByRule[example] = position$
            If $RuleOrder.exampleClassifiedOK[example] = true$
              Decrease $RuleOrder.accuracy$ by one example
            EndIf
          EndIf
        EndIf
EndFor
Output : $RuleOrder$
```

Figure 5: Pseudo-code of the main body of SX

```
Procedure SmartCrossoverOperator
Input : $Parents$,$DefaultClass$
$Examples$ = Get Training Set $Rules$ = Extract all rules from $Parents$
$MatchProfiles$ = Generate the match profile of $Rules$ using $Examples$
$BestRules = \emptyset$
$BestAccuracy = 0$
For as many repetitions as $NumRepetitions$
    ($NewRules$,$NewAccuracy$) = RuleSelectionAlgorithm($MatchProfiles$,$Examples$,$DefaultClass$)
    If $NewAccuracy > BestAccuracy$
      $BestRules = NewRules$
      $BestAccuracy = NewAccuracy$
    Else If $NewAccuracy = BestAccuracy$
      If $NewRules.Length < BestRules.Length$
        $BestRules = NewRules$
        $BestAccuracy = NewAccuracy$
      EndIf
    EndIf
EndFor
$Offspring$ = Generate individual from $BestRules$
Output : $Offspring$
```

The experimentation settings used in these tests are detailed in next section. From table 1 we observe how the new operator achieves a maximum performance boost of 2.4% over CX for both datasets, showing clearly the benefits of SX.

However, a very different situation appears when we test the operator on the *wpbc* dataset. As table 2 shows, SX is never able to outperform CX. Moreover, in one of the configurations the performance decrease achieves 6.1%, which is a quite large performance drop. When looking for the causes of this issue, we observed that the training accuracy of the worst smart crossover configuration had increased by 5.8% over CX, while the average number of rules had doubled. These observations indicate that the cause of this performance drop is overlearning.

In order to fix the problem of overlearning we prune the rule sets generated by SX by removing rules that match only a few examples, as these very specific rules are the main candidates to obtain poor test accuracy. How many examples do we set up as a threshold to apply this pruning process? The aim of this research is to produce an operator that has robust performance across a broad range of datasets in many different circumnstances. For this paper we propose a certain heuristic that has produced reasonably robust results across several datasets. As any heuristic, it has some limitations due to the bias it introduces, but we leave the proposal of a more sophisticated pruning mechanism for further work. The current heuristic works as follows:

1. The heuristic defines a certain threshold. Rules that

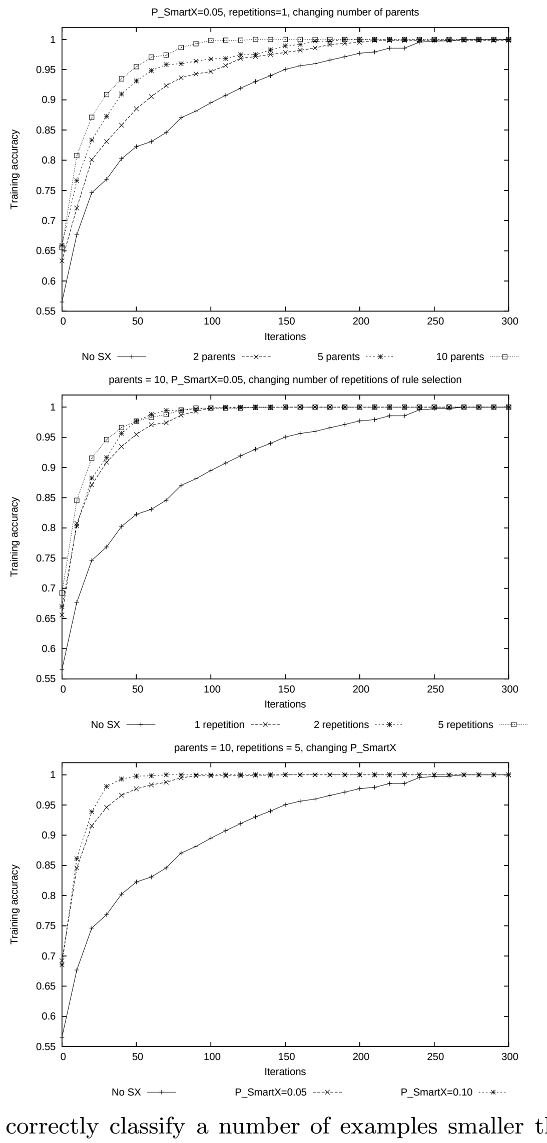
Table 1: **Performance of the original SX on the bal and zoo datasets**

Dataset	Repetitions	$P_{SmartX} = 0.05$			$P_{SmartX} = 0.10$		
		2 parents	5 parents	10 parents	2 parents	5 parents	10 parents
bal	—	Accuracy without smart crossover : 79.0±4.0					
	1	78.8±4.2	80.7±4.3	81.5±3.9	78.6±4.0	80.8±4.0	81.1±3.8
	2	78.6±4.1	80.4±4.1	81.4±3.7	78.4±3.9	80.7±3.7	81.3±3.8
	3	78.5±4.2	80.4±4.2	81.2±3.8	78.1±4.3	80.9±3.8	81.2±3.6
zoo	—	Accuracy without smart crossover : 92.1±8.0					
	1	92.9±7.6	93.6±6.6	94.0±6.7	93.1±7.0	94.3±6.4	94.4±6.7
	2	93.5±7.0	94.1±6.7	94.5±6.6	93.7±6.5	94.1±6.7	94.4±6.3
	3	93.7±7.1	94.5±6.3	94.4±6.4	93.7±6.9	94.2±6.4	94.4±6.4

Table 2: **Performance of the original SX on the wpbc dataset**

Dataset	Repetitions	$P_{SmartX} = 0.05$			$P_{SmartX} = 0.10$		
		2 parents	5 parents	10 parents	2 parents	5 parents	10 parents
bal	—	Accuracy without smart crossover : 75.3±8.3					
	1	74.4±8.4	73.5±9.6	70.8±9.9	74.7±8.5	72.5±10.2	69.2±10.2
	2	74.6±9.5	73.5±8.8	70.5±10.0	74.6±8.5	72.0±9.1	70.4±10.2
	3	74.6±9.5	73.5±8.8	70.5±10.0	74.6±8.5	72.0±9.1	70.4±10.2

Figure 6: Evolution of the training accuracy for the MX-11 dataset under different combination of parameters for SX



correctly classify a number of examples smaller than the threshold are be removed

2. A different threshold is defined for each class

3. The threshold is set at 5% of the training examples belonging to that class

4. The threshold has a minimum value of 5

5. Finally, in order to classify correctly extremely small datasets, the threshold is never higher than 20% of the training examples belonging to that class

The results reported in next section use all the algorithmic considerations described so far.

5 Results

In this section we test SX including the rule pruning code on a set of 25 datasets that represent a broad range of domains in respect to number of attributes, instances, type, ... These problems were taken from the University of California at Irvine (UCI) repository [3], and their features are summarized in table 3. The datasets are partitioned using the standard stratified ten-fold cross-validation method. Also, three different sets of 10-cv folds and 15 random seed are used. This means that the results for each dataset and configuration are the average of 450 runs. Student t-tests are used in order to analyze and compare the performance of SX, using a confidence interval of 95%. The t-tests are applied separately for each dataset. Also, the Bonferroni correction is used for multiple pair-wise comparisons. The used parameters of the system are the ones defined in [2].

For all 25 datasets we test the 18 combinations defined in previous section for the three parameters of the operator. First we will analyze the global performance of each of the 18 configurations for all datasets in order to select the best configuration, and later we will compare, dataset by dataset, this configuration against the original operator.

Table 4 reports the average accuracy over the 25 datasets of each of the 18 configurations. This table is complemented with the results of the t-tests in table 5. We report the t-tests applied between each of the 18 configurations of SX and CX. In average, all 18 tested configurations achieve better performance than CX. The maximum average accuracy difference is 0.4% obtained by the configurations with $P_{SmartX} = 0.10$, number of parents = 10 and repetitions = 2,5. Moreover, only one configuration, and in a single dataset was significantly outperformed by CX, and two configurations managed to outperform significantly the base configuration in four datasets. Based on the combination of the t-tests and the obtained average accuracies, we select the configuration with $P_{SmartX} = 0.10$, number of parents

Table 3: 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%

= 10 and repetitions = 5 to compare it in detail against CX. From table 4 we can identify some general trends. The most sensitive parameter of the operator is the number of parents used in SX, followed by P_{SmartX} and being the number of repetitions of the rule selection algorithm the least important parameter.

Table 6 contains the comparison of several metrics between CX and the selected configuration of SX. We report three metrics: the test accuracy, the average rule-set size of the generated solutions and the algorithm run-time. We can observe how SX obtains better accuracy than CX in 16 of the 25 datasets. In 5 of these datasets the difference is higher than 1%. On the other hand, the maximum performance hit of the new operator compared to CX is 0.64%. The run-time of SX is clearly higher than the run-time of the classic operator. This issue was expected, as this configuration is the one using the highest number of rules to generate one offspring.

The comparison of the number of rules generated by both crossover operators is not uniform. In some cases, such as the gls dataset, SX generates more rules than CX, but these added rules mean a performance boost of more than 3%. On the other hand, on some datasets such as bre, col or vot, SX manages to generate more compact rule sets with higher accuracy, indicating that the algorithm has been able to select the appropriate and minimum number of rules to generate an accurate rule set. However, we can observe how, for the bal dataset, the achieved accuracy is less than the one achieved in previous section by SX without rule pruning. The developed pruning heuristic, although has showed to be quite robust, still has some limitations. Finally, as a reference, the accuracy for several alternative machine learning systems using same the datasets and cross-validation partitions used in this paper is reported in [1].

6 Conclusions and further work

In this paper we have proposed an smart recombination operators designed for the Pittsburgh approach of Learning Classifier Systems. In comparison with some recent Michigan LCS smart recombination approaches, the focus of the new operator does not take place at a rule level, but as a rule set level. In this way we exploit the main characteristic of the Pittsburgh LCS: having a global view of the whole solution. The tested SX selects several parents from the population, evaluates all their rules on all examples and then selects the minimum subset of rules that maximizes the training accuracy of the whole rule set. This selection process has been extended after some short tests with a rule pruning method to avoid the danger of overlearning. The tests performed over a broad range of datasets indicate that the new operator can help GAssist in outperforming significantly the classic crossover operator in several datasets, and it has been never outperformed. Also, in some datasets the final rule-sets obtained were smaller, showing the capacity of SX for avoiding unnecessary or incorrect rules. All this performance boost and robustness has some cost, which is the longer run-time of the system.

There are several lines of future work. The most immediate one is the proposal of a better rule pruning heuristic, as we have observed how the current pruning code, although being quite robust, can compromise part of the possible achievable performance boost. After this improvement, it will be the moment to compare the performance of SX against other paradigms of LCS and other alternative machine learning systems. Moreover, GAssist has recently [14] been applied to very large bioinformatics datasets, of approx 260000 instances. It is important to evaluate the performance of SX on such datasets, and to check if it is feasible to combine successfully SX with the windowing mechanism used in [14] which guarantees a reasonably efficient learning time on these datasets. Alleviating the run-time of the SX operator would also be useful.

Finally, there is another area of smart recombination that we have not treated so far, which acts at the rule level. The proposed operator can only act over the currently existing rules in the population, and in some cases it might be possible that the system is unable to find better individual rules with the current mechanisms. A very interesting future line of research would be to combine both kinds of smart exploration.

7 Acknowledgements

We acknowledge the support provided by the UK Engineering and Physical Sciences Research Council (EPSRC) under grants GR/T07534/01 and GR/62052/01 and the Biotechnology and Biological Sciences Research Council (BBSRC) under grant BB/C511764/1.

8 References

[1] 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. http://www.cs.nott.ac.uk/ jqb/publications/thesis.pdf.

[2] J. Bacardit. Analysis of the initialization stage of a pittsburgh approach learning classifier system. In *GECCO 2005: Proceedings of the Genetic and*

Table 4: Average performance of SX with rule pruning over 25 real datasets from the UCI repository

Dataset	Repetitions	$P_{SmartX} = 0.05$			$P_{SmartX} = 0.10$		
		2 parents	5 parents	10 parents	2 parents	5 parents	10 parents
bal	—	Accuracy without smart crossover : 82.5±13.7					
	1	82.7±13.6	82.8±13.6	82.7±13.5	82.7±13.6	82.8±13.6	82.8±13.5
	2	82.6±13.6	82.7±13.6	82.8±13.6	82.7±13.6	82.8±13.5	82.9±13.4
	3	82.6±13.7	82.8±13.5	82.8±13.5	82.6±13.7	82.8±13.4	82.9±13.4

Table 5: T-tests with 95% confidence level applied to the results of table 4: number of time each conf. of SX outperforms/is outperformed by CX in one of the datasets

Dataset	Repetitions	$P_{SmartX} = 0.05$			$P_{SmartX} = 0.10$		
		2 parents	5 parents	10 parents	2 parents	5 parents	10 parents
	1	0/0	1/0	2/0	1/0	1/0	4/0
	2	0/0	1/0	1/0	1/0	2/0	3/0
	3	0/0	1/0	1/0	1/1	3/0	4/0

Table 6: Comparison of the best configuration for SX against CX

Dataset	Without SX			With SX		
	Test acc.	# rules	#run-time (s)	Test acc.	# rules	#run-time (s)
bal	79.0±4.0	9.9±1.7	16.2±1.8	79.6±4.2	13.7±2.8	61.7±7.2
bpa	62.4±7.8	8.5±1.5	17.7±1.8	63.4±8.0	9.0±1.2	36.1±2.0
bre	70.5±7.9	10.4±1.5	16.3±3.3	71.5±7.6	9.6±1.5	46.5±5.2
cmc	54.4±3.9	8.0±2.0	34.2±2.2	54.1±4.1	14.4±2.8	149.7±27.3
col	93.3±4.3	7.1±1.3	43.4±5.1	93.8±4.4	6.7±1.1	95.9±5.9
cr-a	85.1±4.1	6.6±1.1	30.5±1.8	85.4±3.8	6.4±1.1	69.2±3.4
gls	66.8±9.5	7.2±1.1	38.2±1.3	70.0±9.3	8.8±1.6	89.7±3.1
h-c1	80.4±5.9	8.1±1.2	20.7±1.6	79.9±5.9	8.3±1.5	43.5±1.9
h-h	95.7±3.4	4.6±0.8	19.6±2.5	95.8±3.3	4.6±0.8	40.7±2.8
h-s	80.2±7.6	7.4±1.2	14.6±0.9	80.4±7.2	7.4±1.1	25.9±1.1
hep	89.8±8.0	4.9±0.7	5.9±0.4	90.6±7.5	4.7±0.7	9.9±0.6
ion	92.0±5.2	3.0±1.4	29.8±3.8	91.7±5.2	4.2±1.8	56.8±6.1
irs	95.3±5.6	3.8±0.6	2.3±0.1	94.9±5.8	3.9±0.7	5.1±0.3
lab	98.1±5.4	3.0±0.0	3.8±0.3	97.6±6.0	3.0±0.0	5.3±0.2
lym	80.8±11.2	6.7±1.0	10.8±0.5	82.7±10.2	7.1±1.2	21.3±1.2
pim	74.7±4.8	8.5±1.8	38.6±2.5	74.1±5.2	9.2±1.8	97.8±10.9
prt	47.5±6.7	11.6±1.4	20.5±0.5	49.1±7.0	11.8±1.4	85.2±1.6
son	76.6±9.3	7.5±1.4	68.6±2.8	76.8±8.6	6.7±1.0	102.6±4.7
thy	92.0±5.7	5.1±0.8	3.4±0.2	92.1±5.6	5.4±0.8	7.4±0.5
vot	97.1±3.3	5.8±0.7	3.8±0.2	97.2±3.0	5.7±0.6	11.8±0.7
wbcd	96.1±2.5	2.9±0.6	6.0±0.3	96.0±2.4	3.3±0.7	15.5±1.5
wdbc	94.3±3.1	4.6±0.8	25.3±1.8	94.0±2.9	5.6±0.9	51.2±3.6
wine	93.4±5.5	3.9±0.7	6.8±0.4	93.5±5.3	3.9±0.7	12.0±1.0
wpbc	75.3±8.3	4.1±0.8	11.6±1.0	74.7±9.0	4.0±0.8	16.1±2.5
zoo	92.1±8.0	7.3±0.7	1.7±0.0	94.7±6.1	7.2±0.4	5.6±0.2
ave	82.5±13.7	6.4±2.4	19.6±15.8	82.9±13.4	7.0±3.0	46.5±38.2

Evolutionary Computation Conference, volume 2, pages 1843–1850. ACM Press, 2005.

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

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

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

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

[7] J. J. Grefenstette. Lamarckian learning in multi-agent environments. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 303–310. Morgan Kaufmann, 1991.

[8] G. Harik. Linkage learning via probabilistic modeling in the ecga, 1999.

[9] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. *IEEE-EC*, 3(4):287, November 1999.

[10] P. Larranaga and J. Lozano, editors. *Estimation of Distribution Algorithms, A New Tool for Evolutionnary Computation.* Genetic Algorithms and Evolutionnary Computation. Kluwer Academic Publishers, 2002.

[11] X. Llora, K. Sastry, and D. Goldberg. The compact classifier system: Scalability analysis and first results. In *Proceedings of the Congress on Evolutionary Computation 2005*, volume 1, pages 596–603. IEEE Press, 2005.

[12] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume I, pages 525–532. Morgan Kaufmann, 1999.

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

[14] M. Stout, J. Bacardit, J. Hirst, N. Krasnogor, and J. Blazewicz. From hp lattice models to real proteins: coordination number prediction using learning classifier systems. In *4th European Workshop on Evolutionary Computation and Machine Learning in Bioinformatics 2006 (to appear)*, 2006.

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