Design of an Algorithm based on the Estimation of Distributions to generate new rules in the XCS classifier system.

Joaquín Rivera Padrón [1]
Roberto Santana [2]

Abstract

In classifier systems the genetic algorithms (GAs) have been usually employed as the discovery component. The theory of evolutionary algorithms has achieved important results nowadays, but classifier systems do not seem to be employing these advances in their own benefit. The aim of this paper is to analyze the effect of replacing the traditional discovery component of the XCS classifier system by another kind of population based search method, an Estimation Distribution Algorithm (EDA). The algorithm, which we have called CS-EDA required the implementation of a mutation-like effect with a selected mutation rate. To achieve a proper performance of XCS a new rule deletion method was developed. A more elaborated technique for the calculation of the predictions of the offspring was devised. Finally, , to obtain a categorical comparison between both evolutionary algorithms it was necessary to define performance measures that permitted us to verify that the proposed algorithm performed better than the GA for the examples considered.

Keywords: Classifier systems, XCS, rules deletion method, EDA, genetic algorithm.

MSC: 68T05, 05C35, 05C78, 05C85

1. Introduction.

Since Holland and Reitman [Holland and Rietman, 1978] published their CS-1 a great amount of studies concerning classifiers systems (CSs) have appeared. The needs of studying crucial matters in the functioning of CSs have determined that researchers paid little attention to the evolutionary component of these systems. It is an aim of this paper to make a detailed study of this important component of the classifier systems, particularly the XCS classifier system.

The GAs are the algorithms usually employed as evolutionary component in the CSs. Nevertheless, for the optimization of mathematical functions, alternative population based search methods have recently been proposed [Mühlenbein et al.,1998] [Pelikan et al.,1999]. These algorithms take advantage of the statistical information contained in the population of selected points to improve the search by sampling points in the promising regions of the space of solutions. The Estimation Distribution Algorithms [Mühlenbein,1998] belong to this class of statistical algorithms, and for a number of functions [Mühlenbein et al.,1998] they had shown better global performance than GAs in the optimization.

[1] joaquin@dtz.minaz.cu
[2] Rsantana@cidet.icmf.inf.cu

Institute of Cybernetics, Mathematics and Physics (ICIMAF)
Calle 15 e/ C y D, Vedado, Habana, CP-10400, Cuba

The XCS of Wilson [Wilson, 1995] behaves in an optimal manner in solving difficult problems and has settled theoretical basis for the study of generalizations. The aim of this paper is to analyze the effect of replacing the traditional discovery system of XCS (Genetic Algorithms) by another kind of population based search method (an Estimation Distribution Algorithm). To perform this analysis, a number of statistical measures has been designed that help to understand how the behavior of the Evolutionary Algorithms (EAs) influence the performance of the XCS. Once a proper knowledge about the functioning of the EAs is obtained, the two algorithms are run together in a number of experiments and by observing the statistical measures, we conclude the superiority of the introduced approach.

The paper is organized as follows. Section 2 briefly overviews XCS according to Wilson [Wilson, 1995]. Section 3 early presents the environment employed in the experiments. The discovery component of the XCS (CS-EDA) is discussed in detail in section 4. Section 5 introduces the measures defined to compare the two evolutionary algorithms performances. The experiments are presented and discussed in section 6. Finally, section 7 concludes the paper, summarizing the results obtained.

2. Description of the XCS.

The description of XCS given in this section can be detailed in [Kovacs, 1996] [Padrón and Santana, 1999]. For the interest reader it may be important to revise [Wilson, 1998, 1999] [Lanzy, 1997b, 1997c] for latest research on the XCS.

The rules in XCS store three main parameters: the prediction p_j, the prediction error ε_j, and the fitness F_j. The prediction parameter means the reward the rule expects to gain. The prediction error is an estimate of the error present in the prediction of the rule. The fitness evaluates how precise is the prediction of the rule, and it is a function of the prediction error.

Every time an input signal is sent by the environment to the system, a match set [M] is formed with the rules in the population [P] whose condition part match the input signal. In case the match set [M] is empty a new rule that matches the input is created through an operator called *covering*. Then a system prediction $P(a_i)$ is computed for every action a_i in [M]. This system prediction is used to select the action to be carried out by the system. The selection can be made stochastically or deterministic. Once the selection is made, an action set [A] is formed with the rules in [M] advocating the selected action a_i (rules with its action part equal to a_i). The explore / exploit strategy employed in the experiments of

this work is simply the alternation of one *pure explore problem* [Wilson,1995] with one *pure exploit problem*. For a detailed discussion of explore / exploit strategies revise [Wilson,1996].

Then the selected action (system action) is performed. As a result the environment sends a new input signal together with an immediate reward (r_{imm}). This reward is used to adjust the parameters in $[A]_{-1}$, the action set of the previous time step.

In the XCS classifier system the well-known bucket-brigade method [Goldberg,1989] is not employed to adjust the parameters of the rules. Instead, the adjustments are made using a Q-learning-like technique [Wilson,1994] that has some similarities with the bucket-brigade. The Q-learning-like technique is used to compute a quantity P that is the sum of the reward received in the previous time step and the maximum system prediction in [M] discounted by a discounting factor $\gamma(0 \leq \gamma < 1)$. P is used to update the prediction using the Widrow-Hoff delta rule with learning rate $\beta(0 \leq \beta < 1)$, $p_j \leftarrow p_j + \beta(P - p_j)$. Using this delta rule the prediction error ε_j is updated by the formula $\varepsilon_j \leftarrow \varepsilon_j + (|P - p_j| - \varepsilon_j)$. The fitness calculation is more complex, three steps are required to update it. First, the prediction error is used to compute the accuracy of the rule, $\kappa_j = 0.1 * e^{\frac{\log \alpha (\varepsilon_j - \varepsilon_0)}{\varepsilon_0}}$ if $\varepsilon_j > \varepsilon_0$, and $\kappa_j = 1$ otherwise. The second step is the calculation of the relative accuracy κ'_j of the rule with respect to the rules in its niche. Finally the fitness is updated using the Widrow-Hoff delta rule: $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

The discovery component (GAs, usually) in XCS is performed in niches (action sets). It generates two new rules (a too strong genetic pressure [Lanzy,1997a] can compromise the performance of the XCS) based on the rules present in the niche. Every time an offspring is going to be inserted in the population, the rules in the niche are checked out to see if the condition of any of this rules contains the condition of the new generated rule. If so, the new rule is not inserted, instead a numerosity counter of the old rule is increased. Besides the old rule F_j, p_j and ε_j should have been adjusted more times than a threshold ξ. ξ is called the subsumption threshold.

This technique is known as subsumption deletion, and its application allows the insertion in [P] of only those rules that are more general than the existing ones. The rules numerosity “at born” is one. The rules with numerosity greater than one are known as macroclassifiers. The macroclassifiers are an implementation technique employed to speed up the system, reducing the number of real classifiers the system has to deal with.

The use of macroclassifiers together with the idea of fitness, theoretically results in rules mapping accurately the environment, and in niches having one dominating rule. This effect and its cause, are clearly explained in the generalization hypothesis of Wilson [Wilson,1998]. The interest reader may find a deep study of the generalization capabilities

of XCS in [Lanzy, 1997c]. A more complete review of articles concerning different issues of the XCS may be revised in [Padrón and Santana, 1999].

The deletion method employed.

According to [Wilson, 1997] the deletion method employed in the former XCS article [Wilson, 1995] presented problems in the deletion of low fitnessed classifiers. The deletion method was too "gentler" with the bad rules when there were macroclasifiers in [P].

In our former experiments (not reported here) we noticed the bad influence of an improper functioning of the deletion method in the system performance. This fact determined we devised a novel deletion method that behaves correctly. The reader interested in issues concerning the deletion method may revise [Kovacs, 1999].
The deletion method we devised is intended to function accomplishing the following goals:

- Maintain in [P] high fitnessed rules.
- Delete from [P] low fitnessed rules.
- Maintain niches in [P] that contain a small number of rules.
- Give opportunities (life opportunities) to new created rules to become accurate.

The deletion method firstly assigns a deletion parameter *d* to the rules. The deletion probability of one rule is calculated normalizing its parameter *d* by the sum of the parameters *d* of all the rules.

To behave according to these goals, our deletion method performs the following assignment of *d*:

- If the main parameters of the rule have been adjusted less than νtimes, the parameter *d* will be equal to a minimum value υ
- If the main parameters of the rule have been adjusted more than νtimes, *d* is assigned a value equal to the niche size estimation of the rule divided by its fitness.
- If the numerosity of the rule is greater than $\varphi$the parameter *d* will be multiplied by a factor ϖ

A particular case is when the fitness of the rule is zero, in this case we set the denominator of the division of the second item as the minimum fitness (greater than zero) of the entire population of rules.

A proper setting of the deletion method parameters constitutes a main concern, because an incorrect assignment greatly affects the XCS performance. For instance, if $\nu$is close to ξ the effect will be deleting at random the rules from [P] while they are not accurate. Otherwise, if $\nu$is small (e.g. ν= 2) the newly created rules will not be protected enough. If ω is small the method will have a strong bias to delete accurate rules; instead of, if $\omega$is high

the niches (action sets) of [P] will contain many rules, which could result in the domination of [P] by the former created niches. It is also important to ensure that the choice of υguarantee to the newly created rules the smallest *d*. This is only a theoretical analysis, there were no experiments to determine an optimal setting of the parameters, we assigned values to the parameters on an empirical base.

As will be shown in section 6 this deletion method behaves in a proper manner. Details can be found in [Padron and Santana,1999].

3. The environment for experiments.

The environment employed in this paper is an animat one, the woods2. This environment is used in [Wilson, 1995, 1998] to test the performance of the XCS. Detailed analysis of the woods2 can be found in [Wilson, 1995]. The binary decoding of blank B is 000, of tree T is 010, of predator animat A is 011, of food F is 100, and food G decodes 101. The condition part of one rule is the decoding of its eight surrounding cells. The first variable in the rule decodes the cell above the animat, and the rest of the variables map the cells taken clockwise from the initial position. The action part indicates the cell where the animat goes, it is a number between 0 and 7 corresponding to the order (from left to right) variables have in the condition part.

The animat in woods2 is supposed to learn to avoid trees and predators, and to reach food. The optimal behavior is reaching food at a 1.7 step average, and it was reproduced in our experiments as shown in the figure 3. The initial position of the animat is a random blank cell. In the woods2 there are only 72 possible (legal) signals from a total of 1 953 125 (5 powered to 8). This determines that, for instance, **FBBBFFBT** is a signal from the total space that does not belong to woods2. The figure 1 shows a fragment of the woods2. Blocks similar to the ones showed repeat in every direction in the environment. The blanks are shown as points in the figure.

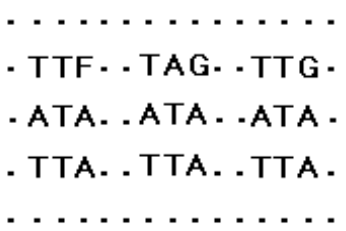

Figure 1. *Blocks of woods2.*

1. fraction = (p_j of the highest fitnessed rule) / 8.
2. iterate through the new rules
   ↳DO p_j = (sumatory of blocks marginals) * fraction.
3. find offspring of highest prediction (new k).
4. d = (p_j of the highest fitnessed rule) - (p_j of new k).
5. add d to pj of every new rule created.

Figure 2. *Predictions calculation with CS-EDA.*

4. The discovery component.

Genetic based optimization methods like GAs have been the evolutionary algorithms usually employed in classifier systems. This paper is the first attempt of applying another type of EA, based on probability distributions, to these systems. In this section we discuss

some necessary steps to design an CS-EDA able to work with a population of XCS-like rules.

In EDA the estimation of the probability distribution of the best individuals is used to sample the points of the next generation, there are no mutation nor crossover operators. The term EDA is also used to group evolutionary computation methods (like GP, GA, etc.) that have in common the use of distribution estimation to model promising solutions and guide the further search [Pelikan et al., 1999].

EDAs can be classified by considering the complexity of the models used to capture the interdependencies between the variables [Larrañaga et al., 1999], simple models consider a lesser number of dependencies among the variables. Other classification can be achieved by grouping EDA according to the way learning is done in the probability graphical model used [Ochoa et al., 1999]. A first class covers the algorithms that make a parametric learning of the probabilities, the second is integrated by those algorithms where a structural learning of the model is done.

The algorithm we use with the XCS for the animat problem (CS-EDA) is a trivariate marginal distribution algorithm that makes a parametric learning of probabilities. It is worth to notice that the codification of the problem addressed by the XCS determines the choice of probabilistic model to be used by CS-EDA.

CS-EDA

STEP 0: Build the population of individuals to evolve from the rules present in $[A]_{-1}$.
STEP 1: Select $M \leq N$ individuals using the truncation selection method. Estimate the distribution $p^s(x,t)$ of the selected rules.
STEP 2: Generate K new rules according to the distribution $p^s(x,t)$.

The truncation selection method employs a parameter T ($0 < T < 1$) that indicates the portion of the population of points to evolve is going to be used to generate new points. In this work $T = 0.25$ in every experiment performed. The new rules are generated using the stochastic universal sampling [Bickle and Thiele,1995].

The EDAs usually do not employ mutation [Mühlenbein et al., 1998], nevertheless it is favorable the effect of the application of this operator in problems where diversity is needed. In the XCS the employment of the discovery component in niches permits the definition of some effect of mutation in the CS-EDA.

Mutation in an EDA.

Mutation is a genetic operator of great importance in the functioning of a major part of the evolutionary algorithms. Mutation incorporates diversity to the population, since it permits the exploration of new regions of the space of solution. In our case, let us suppose

the EA is going to evolve an action set, formed with rules whose condition part match the signal **BBBBBFBB** (000 000 000 000 000 100 000 000). The action set could contain the rules (condition parts) 000 000 000 000 000 100 000 000 and 000 000 000 000 000 10# 000 000, and they will be accurate for the given signal. Generating new rules from them both will never result in rules matching the signal **BBBBBGBB** (000 000 000 000 000 101 000 000), when just a wildcard in the third locus of the sixth block of the rules would solve this difficulty.

New rules that match this signal must be created (by covering), and thus the generalization capability of XCS would be much compromised. The mutation is the operator that permits to cope with this issue. All the above-mentioned determine the need of devising a kind of mutation effect for the CS-EDA (since EDAs do not employ mutation).

In the example above (rules matching **BBBBBFBB**) lets us suppose that the action set contains every rule it can contain. Thus, the sixth block of the rules will be contained in {100, 10#, 1#0, 1##, #00, #0#, ##0, ###}, and so the block 110 will never be the sixth block of none rule of the action set. If happens that 110 is the sixth block of a rule, then this rule will not belong to the action set. This circumstance, which is a consequence of the application of the EA in action sets, permits the developing of a mutation-like effect for the CS-EDA.

In general, the observed circumstance of the impossibility of some blocks to belong to an action set holds for '0', '1' and '#'s in such a way that (in the example above) there will never be a '0' in the first locus of the sixth block of none rule. This fact guaranties the generality (with specific characteristics of the environment to solve, of course) of the mutation-like effect introduced here for the CS-EDA.

In the woods2 the three-genes blocks 111, 11#, 001, and 110 decode none cell value. These blocks can appear in a rule if the mutation is implemented bit to bit. These illegal blocks can be left out of the rules by "mutating" from a three-genes block to another three-genes block. The substitution of one three-genes block by another causes a similar effect, in the phenotype of the new rule, that the substitution of one allele by another does. Thus, the three-genes block mutation will have the same phenotypical impact, in the created rule, that the bit mutation has.

Behind the conception of the mutation-like effect for the CS-EDA is the idea of performing slight alterations in the marginals of the three-genes blocks. Those that conform the rules in the action set and those that are absent (their marginals equal zero). To mutate from a three-genes block to another, the marginals are calculated for the three-genes blocks. In the woods2 there are 27 possible blocks, with four of them illegals (111, 11#, 001, and 110). The mutation effect is achieved in three steps (the following is the calculation made for every one of the eight positions of three-genes blocks). First, the illegal three-genes blocks remain with their marginals to zero. Second, the mutation rate is

divided equally among the legal three-genes blocks absent in that position in the rule. Third, the remaining probability (1 – mutation rate) is shared among the three-genes blocks present in that position in the rule proportional to their marginals.

The latter will result in a kind of heuristic mutation for the CS-EDA. For example, if the sixth block (in the example above) of the rules were 100 and 10#, then their marginals will be 0.5. Let us suppose that the mutation rate is 0.8, then the marginals of the three-genes blocks 111, 11#, 001, and 110 will be zero (so they would never appear in the offspring), and the other 21 three-genes blocks will have their marginals equal to the mutation rate divided by 21.

This way a heuristic (since it restricts the blocks that it generates) mutation-like effect with selected mutation rate for the CS-EDA is achieved. There still remain problems with this mutation effect. It can create illegal rules like 100 000 000 000 000 000 000 000, that matches **FBBBBBBB**, which is not a signal of woods2. In woods2 not every possible signal formed with legal three-genes blocks makes sense. This illegal rule generation can be controlled checking every offspring generated (a very difficult, and little general, solution) or reducing the mutation rate. It is important to notice that the GA also mutates from one three-genes block to another when it is going to mutate.

The offspring prediction calculation method.

In GAs the prediction of the offspring is usually the mean of the parents predictions. In CS-EDA offspring are generated from a set of individuals and not from two parents. In our algorithm assigning to each offspring the same prediction, equal to the mean of all the individuals in the selection set, would not guarantee an appropriate mapping between the genotype of the solution and its prediction value. This reason made necessary to devise a new strategy to estimate the prediction values of the offspring. In our method the main source of information used is the frequency of the alleles of the individuals selected as "parents". The statistical approach we follow permits more sophisticated and exact offspring prediction calculation that can accelerate the desired accurate mapping of the environment, improving the system performance.

The algorithm employed in our work to estimate the offspring prediction aim to accomplish two objectives:

- Differentiate the offspring prediction values according to the frequency of the alleles in the selection pool.
- Adjust the offspring prediction values having as a reference the prediction value of the parents (in this case the rules selected from the selection pool).

After the CS-EDA has been applied and the new rules have been created we accomplish the prediction estimation. To do this we assume that the configuration of each one of the 8

three-genes blocks that define the condition part in the chromosome make an equal contribution to the quality of the prediction. We set the maximum value of this contribution to one eighth of the prediction of the highest fitnessed rule (figure 2, step 1).

For calculating the prediction of each rule, we consider first the frequency of their three-genes blocks in the set of newly created offspring. It is supposed that if a block has a high frequency in the selected set, and thus in the set of generated offspring, is because it is a good building block of the problem. Then the initial prediction of every new created rule will be the sum of the marginals of each one of its eight three-genes blocks multiplied by an eighth of the prediction of the highest fitnessed rule (this is completed in step 2, figure 2).

Finally, we near the prediction of the offspring to the prediction of their parents. This is achieved by calculating the difference between the prediction of the highest fitnessed "parent" and the initial prediction of the offspring with higher initial prediction. This difference is added to the initial prediction of all the new created rules, guaranteeing that the prediction of the offspring with the highest prediction will be the same that the prediction of the best individual in the selected set.

This method complements the identification and mixing of schemata that CS-EDA support. The problem of the offspring prediction calculation is related with the problem of the fitness estimation. Fitness estimation is addressed in Partial Evaluation [Ochoa,1997] and its aim is to reduce the evaluation cost of expensive objective functions . Several solutions [Ochoa and Soto,1997] have been proposed to this latter problem that may be considered for the prediction calculation in the evolutionary component of the XCS.

5. Measures of comparison.

In the aim of comparing this two evolutionary algorithms, and their influence in the performance of XCS, we implemented some measures of performance. These measures (Table 1) are calculated from reports done every time an event of importance in the functioning of XCS occurred. These measures permitted us to have a more detailed idea of the inner functioning of XCS and its evolutionary system, and were of great importance in the final comparison of the performance of both EAs as rule creation methods in the XCS. To complete the analysis, graphical statistics usually employed in literature [Wilson,1995] [Kovacs,1996] are combined with these detailed measures.

A classifier can have different origins: created through covering, created by CS-EDA or created by GA. The parameters (report-parameters) of a rule always reported are its prediction p_j, prediction error ε_j, fitness F_j, accuracy κ_j, lifetime, numerosity, niche average and origin.

Following we summarize the parameters reported in every report:

- whenever a *deletion* occurs: report the report-parameters of the deleted rule.
- whenever any *EA is activated*: report the absolute difference of every offspring prediction to the predictions of its two parents (GA only); absolute difference of every offspring prediction to the predictions of every rule selected (from the selection pool) to participate in the creation of new rules. We also report the origin of every selected rule (from the selection pool) to participate in the creation of new rules.
- whenever a *subsumption deletion* occurs: report the report-parameters of the rule that increments its numerosity, and the total of adjusts performed to the parameters F_j, p_j and ε_j, since the last time this same rule "subsumed" an offspring, or since its creation (if the rule have not "subsum" before).
- whenever a new rule is *inserted* in [P]: report the origin, the p_j, ε_j, and F_j of the rule to be inserted. A detail here is that we do not permit that a rule, that already exists in the population, be reinserted (in case, of course, that the twin rule in [P] can not "subsum" its new copy), to avoid the repetition of a *bad rule*. The latter also obligates the EAs to create rules different from the existing rules in order to insert them in [P]. It is of interest to study the effects in the XCS performance of repeating rules in [P].
- whenever a classifier *equals its ε_j to zero*: report the origin, numerosity, F_j, p_j and total of adjusts the parameters of the rule needed to make its $\varepsilon_j = 0$. It is important to notice that every time a rule is to be inserted in [P] its ε_j is forced to be equal to 1, to monitor the speed of equaling ε_j to 0.
- whenever a rule *equals its κ_j to 1*, or *equals its fitness to 1*: report the same parameters that in the latter case plus the ε_j error of the rules. Besides, in the κ_j case the numerosity is not reported because this parameter will always be 1 at the moment κ_j reaches 1.

With the information contained in the reports the following statistics were calculated:

- *subsumption measures*: the average of the adjusts one rule needs to "subsum" an offspring is calculated by dividing the total amount of adjusts reported in the subsumption deletion reports by the total times a subsumption deletion occurred. This is calculated for the CS-EDA, GA and covering separately. Also the times a rule "subsumed" the two rules created is calculated. These measures permit to know which rules creation method creates macroclasifiers that "subsum" more often.
- *insertion measures*: the total of rules inserted in [P] created through every creation method is accounted. This statistic permits to know which EA inserts more rules in [P], which can occur due to the fact that the rule created is illegal, or it contains more wildcards in its condition than the existing ones in the niche contain (note that this does not imply that the new rule is more general since it can be inaccurate).
- *deletion measures*: this measure gives information about the rules at the moment they are going to be deleted. In the case of microclassifiers we calculate the average of adjusts made to their parameters, average of niche size and average of lifetime of rules. For macroclassifiers the average of their numerosity is calculated. These calculations are performed for rules of different origins separately.

- *pool measures*: the origin of the rules selected (from the selection pool) to participate in the creation of new rules is accounted. This measure permits us to know if any of the EAs creates rules that dominate the selection pools employed in rules creation.
- *prediction measures*: this measures give an idea of how close are the prediction of the new created rules to the prediction of the rules employed in their creation. For every new rule created, an account is augmented every time its prediction equals the prediction of any rule selected from the selection pool; the highest difference of the prediction of the new rules to the predictions of rules selected of the selection pool is calculated. The average of difference of the offspring prediction to the prediction of rules selected of the selection pool is calculated. For rules created by GA every one of these calculations are also made with respect to its two parents.
- *accuracy (κ)* and *prediction error measures*: the average of adjusts needed for rules of each origin to equal its ε_j to 0, and its κ_j to 1 (i.e. become accurate) is calculated. We calculate which method created more rules that were accurate, and the percent this quantity represents in the total of rules created (insertion measures) by this rule creation method. If an EA creates more accurate rules than the other, it is likely to have more rules created by it in [P] and in the selection pools (i.e. better performance) when used both together in the same experiment.
- *fitness measures*: we calculate the average of adjusts needed for rules of each origin to equal its F_j to 1, and the average of the numerosity of the rules when its F_j reaches 1.

6. Experiments and results.

In our work we initially performed experiments (5 runs) with XCS in woods2 with the GA as discovery component and performed experiments (5 runs) with XCS in woods2 with the CS-EDA as discovery component. The figures show clearly that in both cases the optimum performance was reached by the classifier system, but as we expected the graphical statistics used in literature were incapable of offering information to determine which EA performs better than the other. Following we analyze the statistics defined in the last section in order to obtain some knowledge of the inner working of XCS.

The deletion measures showed no superiority of any EA over the other. The calculations of average of numerosity, average of niche size, average of adjusts done to macro and microclassifiers at the time of deletion varied showing no EA better than the other in the experiments performed. The subsumption measures, the fitness measures, and the insertion measures also showed this varying behavior in all the calculations they performed, so they showed no advantage of any EA over the other. It was also noticed that both EAs surpassed the covering in mostly every run in these four measures.

In the prediction measures the rules created by GA showed a lower average of difference to the prediction of rules selected than the rules created by CS-EDA (in every run). The calculation of total of rules created whose prediction equal the prediction of some rule selected of pool was higher for CS-EDA in most of the runs (this is no surprising since the

prediction estimation method employed equals the prediction of one of the rules created to the prediction of the best fitnessed rule). The rules CS-EDA created always showed a higher difference measure than rules created by GA. In the pool measures both EAs were better than covering (which generally only acted at the beginning of the runs) though it is hard to have an idea of the meaning of all these results to compare the performances of the EAs.

The figures referring to the system error, optimum population fitness and entire population fitness (figure 4, 5 and 6) made explicit the good performance of the deletion method employed. The figure 3 shows that in the experiments XCS reached the optimal behavior with every discovery component employed, but still it is difficult to determine superiority of any of the discovery components over the others. The figure 6 shows a close to zero behavior of the system error, indicating an appropriate performance of the system. About figures 3 and 6 it is important to notice that Kovacs also acknowledged the non-asymptotic behavior of the system error and performance for the experiments conducted in (Kovacs,1996); Thus, it is difficult to conclude a judgement from these figures to determine a superior behavior of XCS with any of the discovery components. The figure 4 shows that the fitness average of the macroclassifiers in the population was very close to 1, which is a measure of how precise the macroclassifiers map the environment. Again in figure 4 it is difficult to determine a superior behavior of XCS employing any of the discovery components. The fitness average of the entire population does not support enough information about the existence of categorical differences between the discovery components (figure 5). Nevertheless, the figures above mentioned were very important to observe that in our experiments the XCS with CS-EDA achieves the same behavior that GA when we only look at classical measures of comparisons. It is important to notice that the performance of the system depends in a great amount on the correctness of the deletion method employed.

A total of 10 runs were performed of XCS in woods2 with both CS-EDA and GA as discovery system. The two EAs were alternated one by one every time XCS activated its discovery system. The figures show clearly that the optimum performance was reached by XCS, and also show that none EA created more rules (dominates the population) or created more macroclasifiers than the other. As we expected the subsumption threshold was a strong equalizing factor in the experiments, since it is 20, and rules created by CS-EDA and GA needed at most 3 adjusts to become accurate. Lets analyze the detailed statistics defined here in order to determine a superiority of one of the EAs, if there is any.

Due to the number of factors that are involved in the entire process of deletion the deletion measures are hard to evaluate, so we will only review what we observed. At the time of deletion the rules created by CS-EDA were older (greater lifetime) than the created by GA; also, the rules created by CS-EDA showed a greater numerosity. The parameters of microclassifiers created by CS-EDA were adjusted a lesser number of times than the parameters of microclassifiers created by GA at the moment of being deleted. The number of adjusts performed to macroclassifiers was variable.

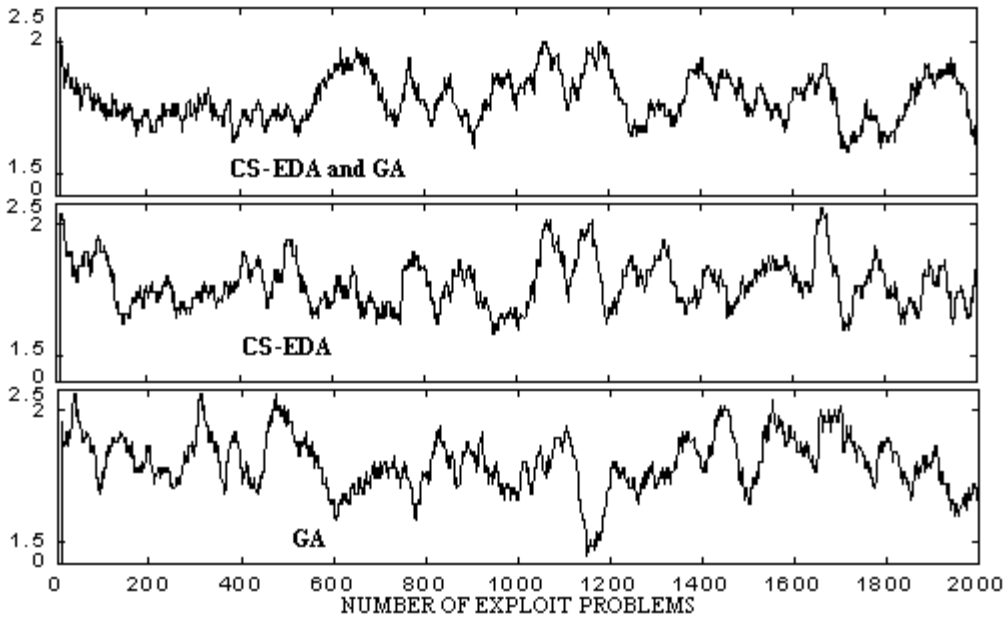

Figure 3. *Performance of the XCS in woods2 in random runs with different discovery components.* $N = 800$, $\beta = 0.2$, $\gamma = 0.71$, $\theta = 25$, $\varepsilon_0 = 0.01$, $\alpha = 0.1$, $\chi = 0.8$, $\mu = 0.01$, $\nu = 7$, $\upsilon = 1$, $\omega = 8$, $\varpi = 100$, $\phi = 0.5$, $P_{\#} = 0.33$, $p_I = 10$, $\varepsilon = 0$, $F_I = 10$, $\xi = 20$, $T = 0.25$.

The prediction error measures showed that a greater number of rules created by CS-EDA made its $\varepsilon = 0$, and showed that the proportion of rules CS-EDA inserted that equals its ε to 0 against the total of rules it inserted in [P] is higher than GA proportion. The accuracy measures showed that a greater number of rules created by CS-EDA became accurate ($\kappa = 1$). Showed also that CS-EDA surpassed GA in proportion of rules inserted in [P] that become accurate against total of rules it inserted and also that the rules needed a lesser number of adjusts (average of adjusts) to become accurate. In our opinion these latter results are the most important in these former experiments.

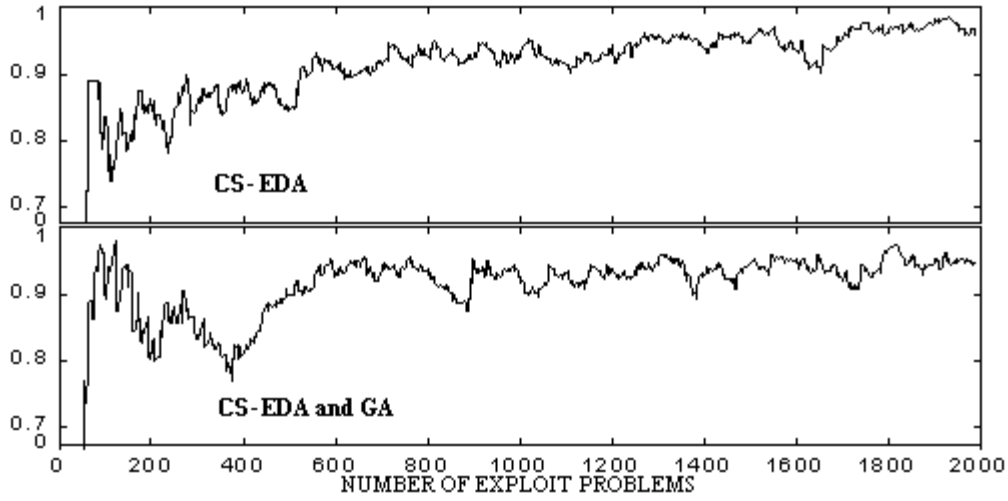

Figure 4. *Fitness average of macroclassifiers in two random selected runs.*

In every run, the CS-EDA surpassed the GA in the subsumption measures. The rules created by CS-EDA needed a number of adjusts to "subsum" lesser than the created by GA needed. The rules created by CS-EDA "subsumed" a number of times greater than the created by AG; rules created by CS-EDA also "subsumed" the two new rules created a number of times greater than the created by AG. Besides the CS-EDA surpassed the GA in velocity of subsumption (i.e. lesser number of adjusts a rule created by CS-EDA needed between subsumption and subsumption). In only one of the runs the GA surpassed the CS-EDA in the total of rules inserted in [P], the rest of the runs the CS-EDA inserted more rules than the GA.

The prediction measures showed the same results that showed in the former experiments with the CS-EDA and the GA used separately. The fitness measure again showed no noticeable differences. The pool measures showed that rules created by CS-EDA participated in the creation of new rules more often than the created by GA, which is an important advantage of the CS-EDA over the GA.

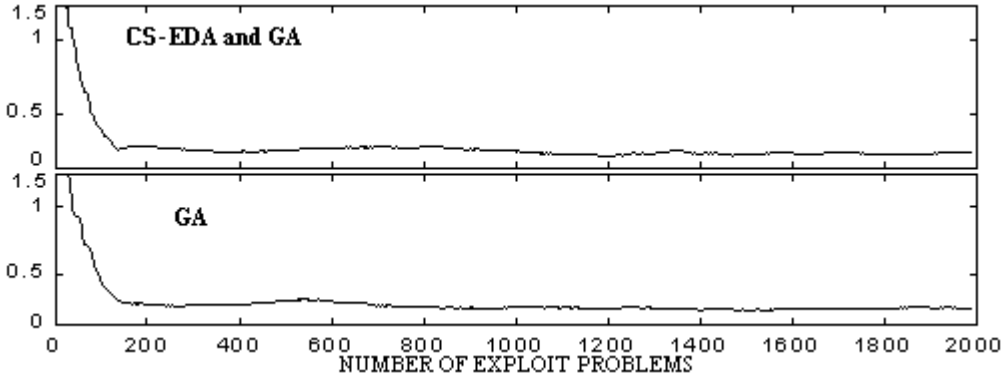

Figure 5. *Fitness average of the entire population in two random selected runs.*

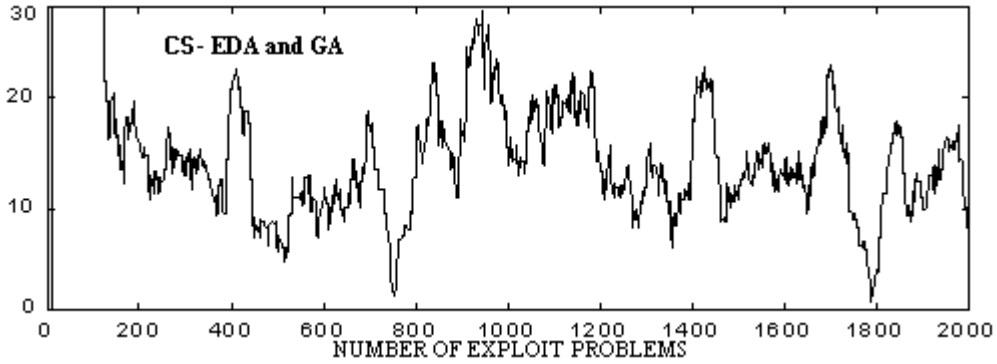

Figure 6. *System error in one random selected run with CS-EDA and GA alternated.*

The accuracy and prediction error measures showed both that CS-EDA was better than GA in the experiments. This means that CS-EDA inserted in [P] a number of rules that became accurate ($\kappa = 1$) greater than GA, and that rules created with CS-EDA needed lesser number of adjusts to become accurate. CS-EDA also surpassed GA in proportion of rules that become accurate against rules inserted. CS-EDA showed a proportion of rules

that equals its prediction error to 0 against the total of rules inserted higher than GA proportion.

	Covering	CS-EDA	GA
Total of times the rules subsumed	638	2534	1304
Total adjusts made to rules between subsumptions	1545	4385	2976
Total of subsumption of two new rules	559	2206	1134
Average of adjusts between subsumptions	2.421630	1.730466	2.282208
Total rules inserted in [P]	2099	19986	20031
Total rules in selection pools	13929	15514	10400
Total rules that make $\varepsilon_j = 0$	0	91	77
Percent of rules that make $\varepsilon_j = 0$ between the inserted	0	0.4553187	0.3844042
Total rules that make $\kappa_j = 1$	7	206	149
Average of adjusts to make $\kappa_j = 1$	46.14286	1.864078	2.020134
Percent of rules that make $\kappa_j = 1$ between the inserted	0.3334921	1.030722	0.743847
Total rules that make Fj = 1	5	31	28
Numerosity until time of deletion	7.789474	7.171779	7.056291
Adjusts done to micros until time of deletion	1.798459	1.124245	1.731215
Adjusts done to macros until time of deletion	8.760129	9.019406	1.065767
Average of the differences to p_j in pool		10.49988	7.755949
Maximum difference to p_j in pool		596.7020	377.29

Table 1. S*ome measures calculated based on reports. XCS in woods2 with CS-EDA and GA together (until problem 2155).*

Again, in our opinion these latter results are the most important in the comparison. Here we summarize it:

- Rules created by CS-EDA become accurate faster than the rules created by GA.
- CS-EDA creates more rules that become accurate, and its proportion of rules that become accurate against rules this algorithm inserted is higher than GA proportion.
- CS-EDA proportion of rules that equals its prediction error to 0 against the total of rules this algorithm inserted is higher than GA proportion.

These results are confirmed in subsumption and pool measures since the first showed that the rules created by CS-EDA "subsum" more often than the created by GA. The second showed that the number of rules created by CS-EDA that participated in the selection pools to create new rules is greater than the number of rules created by GA that participated, and truncation selection takes only the rules of highest fitness. It is also important to notice that *most of the times* CS-EDA creates more diverse rules (since every rule inserted had to be different from the existing ones) than GA, and the three points above show clear that CS-EDA creates "better" rules.

7. Conclusions.

In this paper we employed a CS-EDA as discovery component of the XCS classifier system. It was necessary to design a new rule deletion method for XCS, which proved to solve the problems that the method employed in literature presented. A mutation-like effect was added to CS-EDA. A more elaborated technique to calculate the predictions of the offspring was devised; and new statistical measures that give detailed insights of the inner working of XCS and its discovery system, were employed. A number of experiments were performed in order to test CS-EDA. It was shown that XCS with CS-EDA and XCS with CS-EDA and GA in the environment employed reached its optimum performance. The results shown in section 6 prove that CS-EDA performs better than GA when employed as discovery component of XCS in woods2. More experimentation is the main task at this moment, new environments should be tested, and more difficult problems in which CS-EDA could show its capabilities are necessary.

Bibliography.

Bickle, T.; Thiele, L. (1995). A Comparison of Selection Squemes used in Genetic Algorithms. TIK - Report, Nr 11. Version 2. Second Edition.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley.

Holland, J. H.; Rietman, J. S. (1978). Cognitive systems based on adaptive algorithms.

Kovacs, T. (1996). Evolving optimal populations with XCS classifier systems. Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, U. K.

Lanzy, P. L. (1997). A Study on the Generalization Capabilities of XCS. In Proceedings of the Seventh International Conference on Genetic Algorithms. Morgan Kaufmann.

Lanzy, P. L. (1997). Solving Problems in Partially Observable Environments with Classifier Systems (Experiments on Adding Memory to XCS). Technical Report N. 97.45.

Lanzy, P. L. (1997). A Model of the Environment to Avoid Local Learning (An Analysis of the Generalization Mechanism of XCS). Technical Report N. 97.46.

[Larrañaga et al., 1999] Larrañaga P., Etxechabarria R., Lozano J. A., Sierra B., Inza I., Penna J. M. (1999), A review of the cooperation between evolutionary computation and probabilistic graphical models. Proceedings of the II Second Symposium on Artificial Intelligence. pp 314-324

Kovacs, T. (1999), Deletion Schemes for Classifier Systems. Banzhaf W., Daida J., Eiben A. E., Garzon M. H., Honavar V., Jakiela M. and Smith R. E. (Eds.). Proceedings of the Genetic and Evolutionary Computation Conference GECCO 99. Volume I (pp. 329-336) Orlando Fl. Morgan Kauffman Publishers. San Francisco, California

Mühlenbein, H. (1998). The equation for response to selection and its use for prediction, Evolutionary Computation 5, pp. 303-346.

Mühlenbein, H.; Mahnig, T., Ochoa A. (1998). Squemata, Distributions and Graphical Models in Evolutionary Optimization, to appear in Journal of Heuristics Vol. 5, No. 2. Also at http://set.gmd.de/AS/ga.publineu.html#index.

Ochoa, A. (1997). How to deal with costly fitness functions in evolutionary computation. Proceedings of the 13th ISPE/IEE International Conference on CAD/CAM Robotics & Factories of the Future'97. Universidad Tecnológica de Pereira, diciembre 15--17, 1997, Colombia. (pp: 788--793).

Ochoa, A., Soto, M. R. (1997) Ochoa A., Soto M. R. (1997) . Clustering based data exploration methods for partial evaluation of Genetic Algorithms. In Proceedings of the 13th ISPE /IEE International Conference on CAD/CAM, Robotics and Factories of the Future. pp. 713-717.

Ochoa A., Soto M., Santana R., Madera J. C., Jorge N. (1999). The Factorized Distribution Algorithm and The Junction Tree: A Learning Perspective. Proceedings of the II Second Symposium on Artificial Intelligence CIMAF'99. pp. 368-377.

Padron, J. R.; Santana R. (1999). Use of estimation distribution algorithms as evolutionary component of the XCS classifier system. Diploma Thesis. Faculty of Mathematics, Havana University.

Pelikan, M.; Goldber D. E., Cantú-Paz E.(1998). Linkage problem, Distribution estimation and Bayesian Estimation Networks. Illegal Report 98013. November 1998

Pelikan, M.; Goldber D. E., Lobo F. (1999). A survey of optimization by making and building probabilistic models. Illegal Report 98018. September 1999

Wilson, S. W. (1994). ZCS: A zeroth level Classifier System. Evolutionary Computation, 2 (1). (pp 1--18).

Wilson, S. W. (1995). Classifier fitness based on accuracy. Evolutionary Computation, 3 (2)

Wilson, S. W. (1997). XCS implementation notes. Available at http://netq.rowland.org/sw/swhp.html

Wilson, S. W. (1998). Generalization in the XCS classifier system. From Genetic Programming 1998: Proceedings of the Third Annual Conference, J. Koza et al. (Eds.). San Francisco, CA: Morgan Kaufmann.

Wilson, S. W. (1999). State of the XCS classifier system research. Technical Report 99.1.1, 1999. Available at http://prediction-dynamics.