

Accuracy-based Learning Classifier System Ensembles

with Rule-Sharing

Larry Bull, Matthew Studley, Tony Bagnall* & Ian Whittle*

UWE Learning Classifier Systems Group Technical Report UWELCSG05-009

School of Computer Science

University of the West of England

Bristol BS16 1QY, U.K.

larry.bull@uwe.ac.uk

*School of Computer Sciences

University of East Anglia

Norwich NR4 7TJ, U.K

Abstract

This paper presents an investigation into exploiting the population-based nature of Learning Classifier Systems for their use within highly-parallel systems. In particular, the use of simple accuracy-based Learning Classifier Systems within the ensemble machine approach is examined. Results indicate that inclusion of a rule migration mechanism inspired by Parallel Genetic Algorithms is an effective way to improve learning speed in comparison to an equivalent single system. Presentation of a mechanism which exploits the underlying generalization mechanism of such systems is then shown to further improve performance, particularly as task complexity increases. Finally, considerably better than linear speed-up is demonstrated on a well-known benchmark task.

Keywords: data mining, genetic algorithms, parallel systems, reinforcement learning.

1 Introduction

There is now widespread recognition that it is possible to extract previously unknown knowledge from large datasets using machine learning techniques. As the use of machine learning for exploratory data analysis has increased, so have the sizes of the datasets they must face (Giga and Terabyte datasets are common place) and the sophistication of the algorithms themselves. For this reason there is a growing body of research concerned with the use of parallel computing for data mining (e.g., see [Zaki & Ho, 2000]). We are currently producing a super-computing data mining resource which utilises a number of advanced machine learning and statistical algorithms for large datasets. In particular, a number of Evolutionary Algorithms (EAs) and the Ensemble Machine (EM) approach are being used to exploit the large-scale parallelism possible in super-computing.

One of the most common ways for EAs to model a dataset is through the use of production system rules, i.e., rules of the general form IF <condition> THEN <class>. In these systems, known as Learning Classifier Systems (LCS)[Holland, 1976], either each candidate solution is a complete set of rules [Smith, 1980] or the EA manipulates individual rules within a single rule-base/population (as in [Holland, 1976]). Rules use conjunctive logic for each input variable/feature in combination with implicit feature selection through a generalization mechanism. The utility of LCS to develop appropriate rules for mining tasks was demonstrated in a number of early applications including sequence prediction [Riolo & Robertson, 1988] and health informatics [Bonelli & Parodi, 1991]. Bernado et al. [2002] have shown how the most current versions of the two LCS formalisms – GALE [Llora, 2002] and XCS [Wilson, 1995] respectively – are extremely competitive with a number of well-known machine learning techniques (see also [Bull, 2004] for examples). XCS in particular has demonstrated excellent performance on a number of data mining tasks. For example, Greenver's [2000] XCS was ranked third in the 2000 European Network on Computational Intelligence (COIL) data mining competition, being beaten only by algorithms incorporating problem specific heuristics, and Wilson [2001] used a version of XCS on the Wisconsin breast cancer data to new levels of accuracy, producing rules that highlighted previously unknown features of the data. XCS builds a complete map of the given problem surface. That is, a Genetic Algorithm [Holland, 1975] is used to produce generalizations over the space of all possible condition-class combinations. It has been highlighted that XCS has the ability to create maximally general rules that map the problem space in the production system format using the most efficient rule-set possible [e.g., Butz et al., 2004]. Importantly, this feature allows the user to examine how the LCS has achieved its performance.

Within the wider machine learning community a number of techniques have been proposed for the discovery of hidden relationships between the variables of a given dataset, such as rule induction algorithms (e.g., C4.5)[Quinlan,

1993], artificial neural networks, instance-based algorithms, statistical approaches [e.g., James, 1985], etc. As the amount of data being collected continues to grow at a rapid rate, a number of parallel computing implementations for data mining have been presented to ease the use of such techniques, see for example InfoCharger [George & Knobbe, 1999], SPRINT [Shafer et al., 1996], and Weka-parallel [Celis & Musicant, 2003] (overviews can be found in [Zaki & Ho, 2002][Freitas & Lavington, 1998]). In general such systems are either data parallel, where the data is divided between processors, or task parallel, where portions of the search space are assigned to different processors. In the former, the simultaneous consideration of outputs from more than one algorithm for a given input presented to all algorithms, e.g., by majority voting, has been found to give improved performance over the traditional use of a single algorithm [e.g., Dietterich, 2000]. Versions of this Ensemble Machine approach, particularly non-overlapping initial training data systems, would also appear to be ideally suited for use on a super-computing resource since each algorithm operates independently and so can be executed in parallel on different processors before a final output is determined; with a large number of processors available, the potential use of large ensembles becomes possible for large datasets. We consider the use of LCS within EMs here.

2 A Simple Accuracy-based LCS

In this paper we use a version of the simple accuracy-based LCS termed YCS [Bull, 2005] which is a derivative of XCS. YCS is without internal memory, the rulebase consists of a number (N) of condition/action rules in which the condition is a string of characters from the usual ternary alphabet $\{0,1,\#\}$ and the action is represented by a binary string. Associated with each rule is a predicted payoff value (p), a scalar which indicates the error (ϵ) in the rule's predicted payoff and an estimate of the average size of the niches (action sets - see below) in which that rule participates (σ). The initial random population has these parameters initialized, somewhat arbitrarily, to 10.

On receipt of an input message, the rulebase is scanned, and any rule whose condition matches the message at each position is tagged as a member of the current match set $[M]$. An action is then chosen from those proposed by the members of the match set and all rules proposing the selected action form an action set $[A]$. A version of XCS's explore/exploit action selection scheme will be used here. That is, on one cycle an action is chosen at random and on the following the action with the highest average payoff is chosen deterministically.

The simplest case of immediate reward (payoff P) is considered here. Reinforcement in YCS consists of updating the error, the niche size estimate and then the payoff estimate of each member of the current $[A]$ using the Widrow-Hoff delta rule with learning rate β :

$$\varepsilon_j <- \varepsilon_j + \beta(|P - p_j| - \varepsilon_j) \quad (1)$$

$$\sigma_j <- \sigma_j + \beta(|[A]| - \sigma_j) \quad (2)$$

$$p_j <- p_j + \beta(P - p_j) \quad (3)$$

The original YCS employs two discovery mechanisms, a panmictic (standard global) GA and a covering operator. On each time-step there is a probability g of GA invocation. The GA uses roulette wheel selection to determine two parent rules based on the inverse of their error:

$$f_j = (1 / (\varepsilon_j^\nu + 1)) \quad (4)$$

Here the exponent ν enables control of the fitness pressure within the system by facilitating tuneable fitness separation under fitness proportionate selection (see [Bull, 2005] for discussions). Offspring are produced via mutation (probability μ) and crossover (single point with probability χ), inheriting the parents' parameter values or their average if crossover is invoked. Replacement of existing members of the rulebase uses roulette wheel selection based on estimated niche size. If no rules match on a given time step, then a covering operator is used which creates a rule with the message as its condition (augmented with wildcards at the rate $p_\#$) and a random action, which then replaces an existing member of the rulebase in the usual way. Parameter updating and the GA are not used on exploit trials.

In this paper, to aid the generalization process, the panmictic GA is altered to operate within niches (see [Butz et al., 2004][Bull, 2005] for discussions). The mechanism uses XCS's time-based approach under which each rule maintains a time-stamp of the last system cycle upon which it was part of a GA. The GA is applied within the current [A] when the average number of system cycles since the last GA in the set is over a threshold θ_{GA} . If this condition is met, the GA time-stamp of each rule is set to the current system time, two parents are chosen according to their fitness using standard roulette-wheel selection, and their offspring are potentially crossed and mutated, before being inserted into the rulebase as described above.

YCS is therefore a simple accuracy-based LCS which captures the fundamental characteristics of XCS whilst remaining amenable to close modelling [Bull, 2005]: "[E]ach classifier maintains a prediction of expected payoff, but the classifier's fitness is *not* given by the prediction. Instead the fitness is a separate number based on an inverse function of the classifier's average prediction error" [Wilson, 1995] and a "classifier's deletion probability is set proportional to the [niche] size estimate, which tends to make all [niches] have about the same size, so that classifier resources are

allocated more or less equally to all niches” [ibid.]. The evolutionary algorithm in YCS is a steady-state GA. A simple steady-state GA without genetic operators can be expressed in the form (after [DeJong & Sarma, 1993]):

$$n_j <- n_j + n_j R_j - n_j D_j \quad (5)$$

where n_j refers to the number of individuals of type j in the population, R_j refers to their probability of reproductive selection and D_j to their probability of deletion. Roulette-wheel selection is used in YCS:

$$R_j = f_j / f_{[P]} \quad (6)$$

where f_j is the fitness of individuals of type j (Equation 4) and $f_{[P]}$ is the total population ([P]) fitness. Replacement is proportional to estimated action set size, i.e.:

$$D_j = \sigma_j / \sigma_{[P]} \quad (7)$$

Bull [2002] presented a simple Markov model of the GA in an LCS working within niches, examining the difference in fitness pressure between two rule types - accurate and inaccurate or general and specific. The reproductive bias inherent in more general rules was approximated very simply for such executable models through increasing the fitness of the more general rule by a given factor. A similar approach can be used by altering Equation 4:

$$f_j = \pi_j (1 / (\epsilon_j^v + 1)) \quad (8)$$

where π_j is the proportion of all possible action sets in which the rule participates.

Table 1 shows the payoffs for the single-step task with a single-bit condition and single-bit action (after [Kovacs, 2000], taken from [Bull, 2005]). The last two entries in Table 1 show the expected payoff for the general rules, i.e., the predicted payoff of a general rule is the average of the payoffs it receives (assuming equal probability). It can be seen that under this scheme for input '1' the general rule #:0 has a higher predicted payoff than the correct rule 1:1; #:0 is an overgeneral rule which would cause sub-optimal performance (it is a 'strong' overgeneral [Kovacs, 2001]).

After [Bull & Hurst, 2002], using equations of the general form shown in Equation 5 a very simple model of the expected proportions of each rule type in the next generation can be determined; by specifying the initial proportions of

each rule in the population ($N/6$), it is possible to generate the trajectory of their proportions over succeeding generations. Note partial individuals are allowed and hence it is in effect an infinite population model. The trajectory of their related parameters can also be generated (e.g., see [Bull & Hurst, 2002][Bull, 2005]). In the following it is assumed that both inputs are presented with equal frequency, that both actions are chosen with equal frequency and that the GA fires once every four cycles (i.e., always explore trials). The rules' parameters are updated according to the equations above on each cycle with $\beta=0.2$ and $\nu=1.0$. The progress of all six rules is examined here, with rulebase size $N=400$. Note for the problem in Table 1, the two rules containing generalization have $\pi_j = 2/4$, whereas for the specific rules $\pi_j = 1/4$.

Table 1: Reward payoffs for simple single-step task considered.

Input	Output	Payoff
0	0	3000
0	1	1000
1	0	800
1	1	1000
#	0	1900
#	1	1000

Figure 1 shows the behaviour of the modelled YCS on the single-step task defined in Table 1. It can be seen that the accurate general rule #:1 is the most numerous, with an equal number of the accurate specific rules 1:0 and 0:0. That is, the system has converged upon the maximally general solution to the problem as expected by current understanding of the niche GA within accuracy-based LCS (e.g., [Wilson, 1995],[Butz et al., 2004],[Bull, 2005] – see also Section 6).

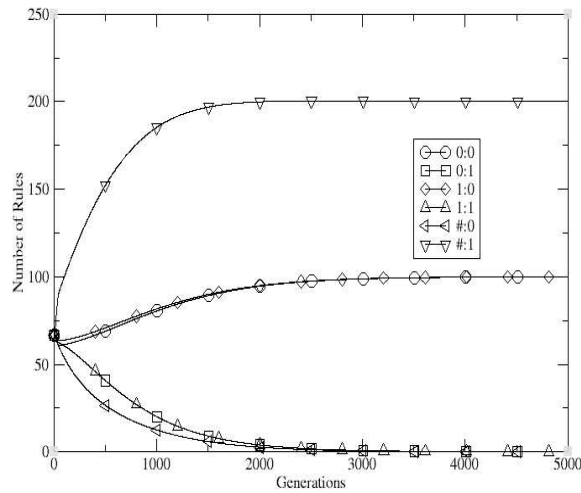


Figure 1: Behaviour of modelled YCS population on the simple problem.

3 The Multiplexer Task

We use versions of the well-known multiplexer task in this paper. These Boolean functions are defined for binary strings of length $l = k + 2^k$ under which the k bits index into the remaining 2^k bits, returning the value of the indexed bit. A correct classification results in a payoff of 1000, otherwise 0.

Figure 2 shows the performance of YCS on the 20-bit multiplexer problem with $N=2000$, $p_{\#}=0.6$, $\mu=0.04$, $\nu=10$, $\chi=0.5$, $\theta_{GA}=25$ and $\beta=0.2$. After [Wilson, 1995], performance from exploit trials only is recorded (fraction of correct responses are shown), using a 50-point running average, averaged over ten runs. It can be seen that optimal performance is reached around 60,000 trials, matching that of XCS [e.g., Butz et al., 2004], with the average error dropping to roughly 10% of the payoff range. Figure 2 also shows the average specificity (fraction of non-# bits in a condition) for the LCS. That is, the amount of generalization produced. The maximally general solution to the 20-bit multiplexer has specificity $5/20 = 0.25$ and YCS can be seen to produce solutions with the same average specificity as this.

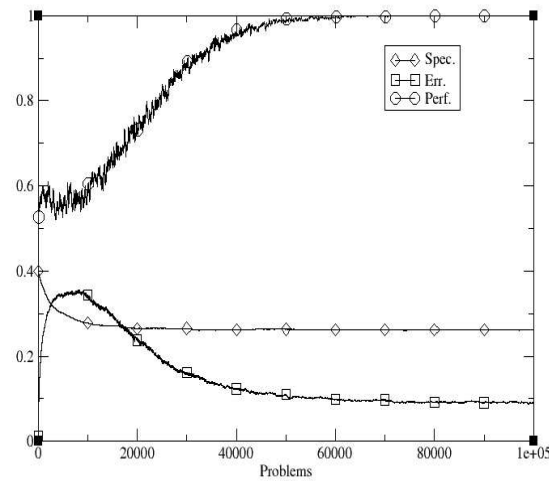


Figure 2: Performance of YCS on the 20mux problem.

4 Ensembles of LCS

Ensemble Machines [e.g., Ho et al., 1994] are machine learning systems which consist of a number of components that each individually produces a solution to a problem. No one model will outperform all others for all queries, hence the principle of ensemble machines is to combine the output of several models to find an overall solution that utilises the strength of the constituents and compensates for their individual weaknesses. This combination of algorithms is often constructed by measuring and maximising the diversity (complementary characteristics) between the individuals, although its predictive power remains unclear (e.g., [Brown et al., 2004]). Perhaps most related to this work, Cantu-Paz and Kamath [2003] have used a simple evolutionary algorithm to successfully design decision trees within an ensemble.

A number of investigators have examined the use of LCS in other multi-system environments. Early examples include Dorigo et al.'s (see [Dorigo & Colombetti, 1998] for an overview) use of multiple LCS within a hierarchical structure to control an autonomous robot and Seredynski et al.'s [1995] examination of reward sharing in a simple iterated game. More recently, Hercog and Fogarty [e.g., 2002] used LCS to represent the agents of a simple game and Bull et al. [e.g., 2004] used LCS to control road-traffic junctions. There has also been some related work within the field of computational economics. After Holland and Miller [1991], a number of researchers have used LCS to represent traders within artificial markets. Early examples include [Marimon et al., 1990] and [Palmer et al., 1994]. More recently,

Bull [e.g., 1999] considered Continuous Double-Auction markets and Bagnall and Smith [e.g., 1999] modelled the energy producers of the UK electricity market.

Figure 3 shows the performance of an ensemble of 10 YCS, each with the same parameters as that shown in Figure 2. Here an explore trial consists of presenting each LCS with a new random input string, with all functionality as described above. An exploit trial consists of presenting all LCS with the same random input string and using a majority classification voting scheme such that, if 5 or more LCS give the correct answer, the trial is considered to have been a success. As can be seen, the number of trials to reach optimality reduces to around 40,000 here, a promising reduction in time of a third over the single LCS above; a ten-fold reduction is, of course, unlikely.

We note that the overall ensemble contains ten times as many rules as the traditional single system. Figure 4 shows how using a rule-base of $N=20,000$ provides the same performance as the ensemble. We now consider ways in which to improve the learning speed of the LCS ensemble beyond that of a single LCS containing the same total rule resource.

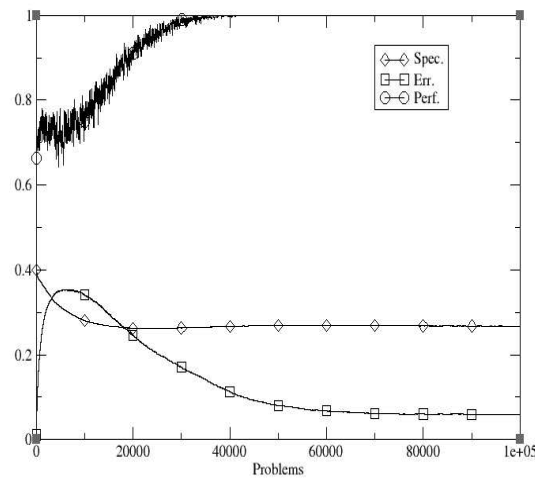


Figure 3: Performance of YCS ensemble of 10 systems on the 20mux problem.

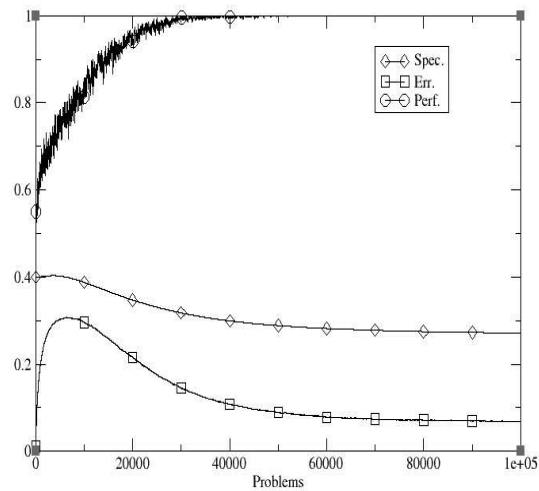


Figure 4: Performance of YCS with $N=20,000$ on the 20mux problem.

5 Rule Sharing in Ensembles

For non-trivial problems the use of a simulated evolutionary process can be very computationally expensive. Therefore approaches have been proposed which enable EAs to exploit parallel computing resources, in particular, those which impose a structure upon the population have been found most beneficial. Such structure can consist of a number of interconnected “islands” - sub-populations in a hypercube - between which individuals are periodically shared [e.g., Cohoon et al., 1987], or a planar grid of processors - one candidate solution per node - across which good solutions diffuse through localised breeding [e.g., Manderick & Spiessens, 1989]. Investigations have shown that performance improvements can also be obtained by such approaches, along with reductions in execution time, due to an increased level of diversity within the global population from the restricted mating schemes [e.g., Cantu-Paz, 2000]. Previous work on the evolution of data miners using a fine-grained EA include [Folino et al., 2000] and with the island approach [Neri & Giordana, 1995].

Given that each YCS consists of a population of co-evolving rules, the island-model approach is analogous to the LCS ensemble described in the previous section. However we note that, with a very large number of locally connected processors, the super-computer scenario is not unlike the fine-grained EAs.

We have explored the inclusion of a rule migration mechanism within the LCS-EM based on the island-model EA. Here, on each explore trial, a probability ϕ is tested within the given LCS. If the probability is satisfied, some

fraction ϕ of the rule-base is chosen (with replacement), based on fitness, to be migrated using the same algorithm as for GA reproduction. A recipient LCS is then chosen at random from the other ensemble members. The recipient inserts the new rules into its rule-base using the same algorithm as for GA deletion. Thus, as in island-model systems, the search process is augmented by the influx of diverse rules selected based on their good (local) fitness to be used in further search by a given LCS, as opposed to relying purely upon the standard stochastic search operators.

The potential benefits of this approach can be considered within the above model by alteration to Equation 5:

$$n_j \leftarrow n_j + n_j R_j - (1 + \phi)(n_j D_j) + \phi M_j \quad (9)$$

where M_j specifies the probability of receiving a copy of an individual of type j from another ensemble member.

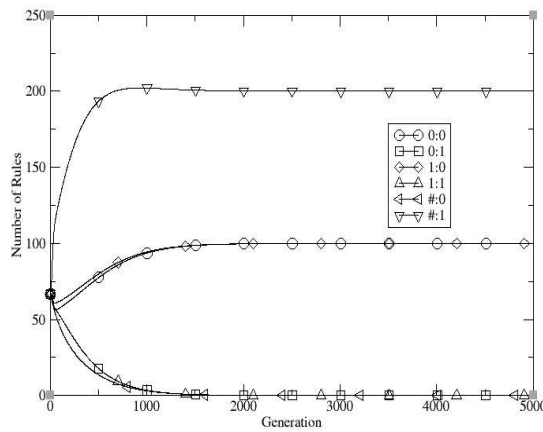


Figure 5: Behaviour of modelled YCS with rule sharing.

Equation 9 makes explicit the changes in deletion probability and numerosity from such a sharing mechanism. Figure 5 shows the behaviour of the modelled YCS in figure 1 with the simple assumption that $M_j = n_j R_j$ and $\phi=1.0$, i.e., two identical LCS form the ensemble considered. As can be seen, the rate of propagation of the maximally general solution is faster than that shown in figure 1; rule migration increases the learning speed.

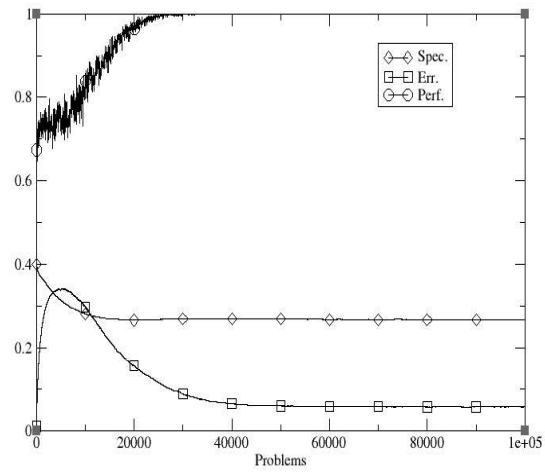


Figure 6(a): Ensemble with low sharing rate and fraction.

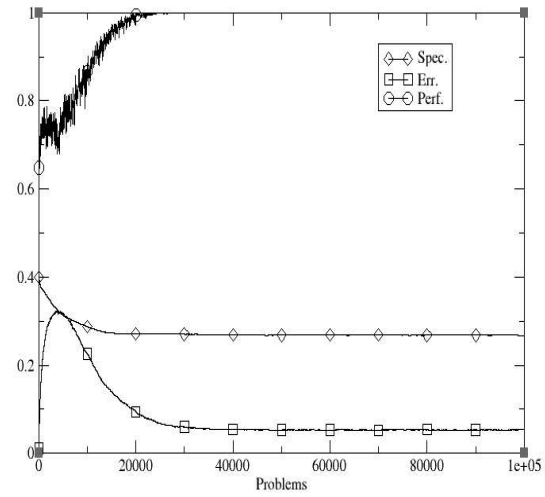


Figure 6(b): Ensemble with low rate and high fraction.

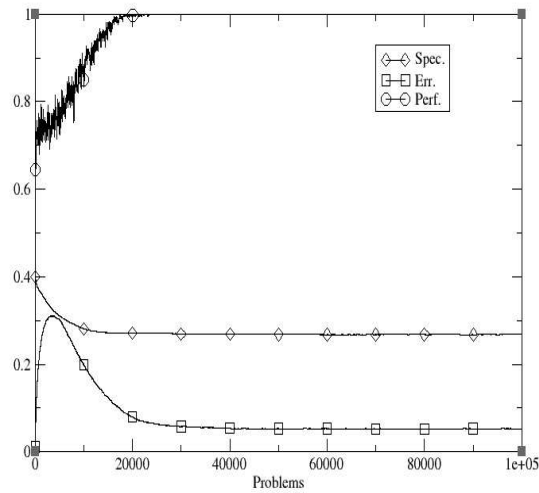


Figure 6(c): Ensemble with high rate and low fraction.

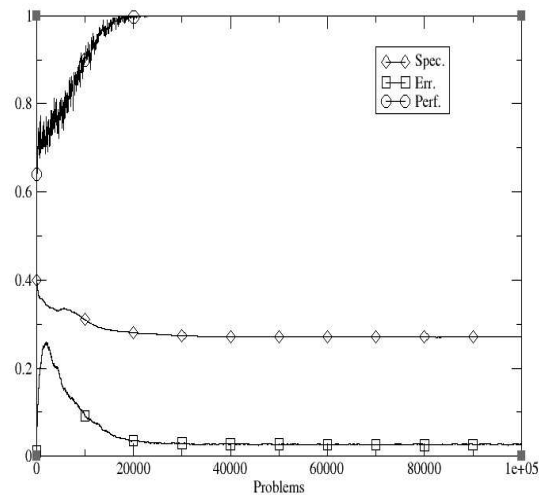


Figure 6(d): Ensemble with high sharing rate and fraction.

Figures 6(a-d) shows the performance of the rule sharing ensemble of 10 YCS on the 20-bit multiplexer problem using the same parameters as before. The effects from varying the rate at which rule sharing occurs and the amount of rules then shared have been examined. Figure 6(a) shows the case with $\varphi=0.001$ and $\phi=1\%$. These two values are taken as “low” here. Similarly, figure 6(d) shows the case with $\varphi=0.01$ and $\phi=10\%$. These two values being taken as “high” here. As can be seen, the reduction in the learning time is about the same in all cases – around 20,000 trials, a reduction in time of two-thirds over the single LCS. Thus rule-sharing is beneficial to the LCS ensemble machine. However there is no difference

in the time taken for the average specificity to approach that of the maximally general solution with any combination of the rule sharing parameters.

Figures 6(a-d) also show how there is a decrease in the average error, particularly when *both* parameters are high. Conversely, empirical results with the island-model GA [e.g., Cahoon et al., 1987] suggest that one parameter should be high, with the other low for maximised benefit from migration events. Average errors rise to around 30%, or 25% in the high parameter case, and drop to 5% or 2.5% of the payoff range. This is in contrast to the average errors dropping to around 10% in the previous experiments.

6 Niche-based Rule Sharing

As noted above, it is well-established [e.g., Butz et al., 2004] that the triggered niche GA creates a significant pressure within an accuracy-based LCS to produce rules which are maximally general. That is, rules which only consider the problem attributes which are predictive for the given task. This is achieved by the fact that more general and accurate rules participate in more niches ([A]) than specific but equally accurate rules, which means the former experience more reproductive opportunities than the latter. Thus rule discovery is niche-based. The previous rule migration mechanism is, like island-model GAs, panmictic; selection for which rules to migrate occurs across the whole rule-base/population. Parallel GAs are known to benefit from a niching affect and the results above indicate this is also true for LCS-EM. However the panmictic scheme requires (potentially) considerable extra processing within each LCS to identify the rules to be migrated.

We have also investigated a niche-based rule migration scheme which draws on the underlying rule discovery mechanism of the simple accuracy-based LCS. Here each rule maintains a time-stamp of the last system cycle upon which it was part of a rule migration event. Migration is applied within the current [A] when the average number of system cycles since the last rule sharing in the set is over a threshold θ_{SH} . If this condition is met, the migration time-stamp of each rule in the given [A] is set to the current system time, a *single* rule is chosen according to their fitness using the standard roulette-wheel selection, before being inserted into the rulebase of another member of the ensemble as described above. Again, this only occurs on explore trials.

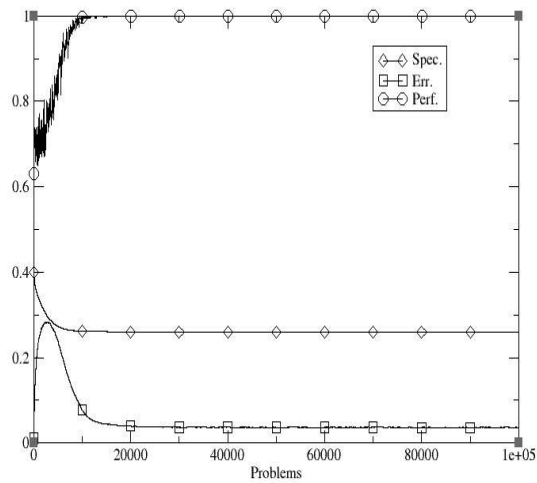


Figure 7(a): Ensemble with $\theta_{SH} = 10$.

Figure 7(a) shows the performance of the ensemble with $\theta_{SH} = 10$. As can be seen, the time taken to reach optimal behaviour is reduced to around 10,000 trials. Experiments with the panmictic scheme indicate that much higher levels of migration, e.g., $\varphi=0.1$, do not give faster performance than above (not shown).

Figure 7(b) shows how, even with a larger value of θ_{SH} (e.g., $\theta_{SH}=100$), roughly the same performance is obtained from this mechanism. We therefore conclude that not only is the scheme less computationally expensive but it is able to exploit the propagation of more general, accurate rules by the triggered niche GA across the LCS ensemble. This scheme would also result in less data being passed inter-processor than under the panmictic scheme.

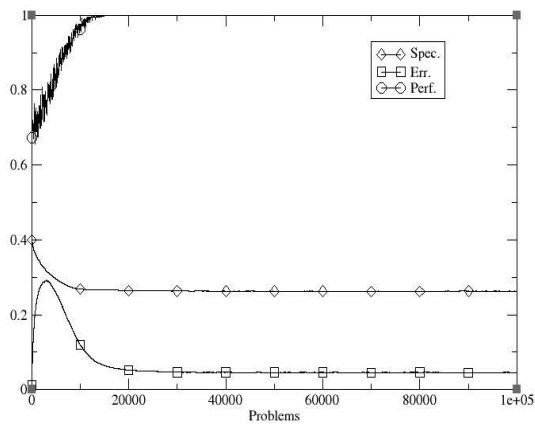


Figure 7(b): Ensemble with $\theta_{SH} = 100$.

Figure 7(c) presents the results from figures 3, i.e., an ensemble of 10 YCS on the 20-bit multiplexer without rule sharing, and 7(a), i.e., the same initial systems but with niche-based sharing during learning. A two-tailed T-test on the explore trial at which the succeeding 50 trials are correct shows the latter is better than the former with high significance ($p < 0.01$).

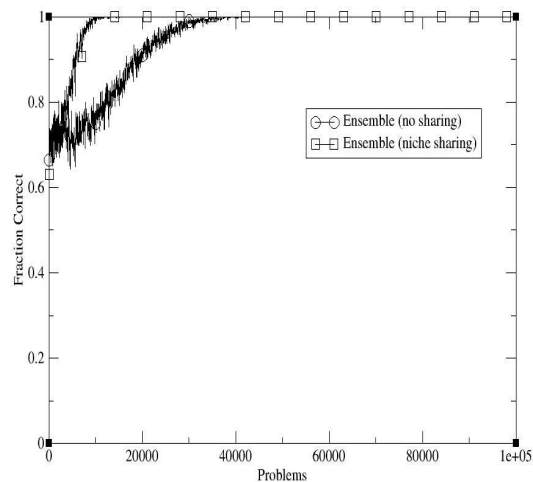


Figure 7(c): Comparison of ensemble performance with and without sharing.

Finally, we have examined the performance of the ensemble using this migration scheme on two larger versions of the multiplexer task. Figure 8 shows the performance of the same system as in figure 7(a) but with $N=5000$ on the 37-bit multiplexer problem. As can be seen, optimal performance is reached around 50,000 trials which can be contrasted with the 750,000 trials taken by a single YCS with the same parameters (not shown - also same for XCS [e.g., Butz et al., 2004]). Figure 9 shows the performance of the same system ($N=5000$) but $p_{\#}=0.75$ on the 70-bit multiplexer problem with optimality reached around 400,000 trials. Butz et al. [2004] show XCS solving the same problem with $N=50,000$ and $p_{\#}=0.75$ around 3,500,000 trials. Thus as task difficulty increases we see an increase in the relative difference in performance obtained through the rule migration scheme; *the speed-up is better than linear here.*

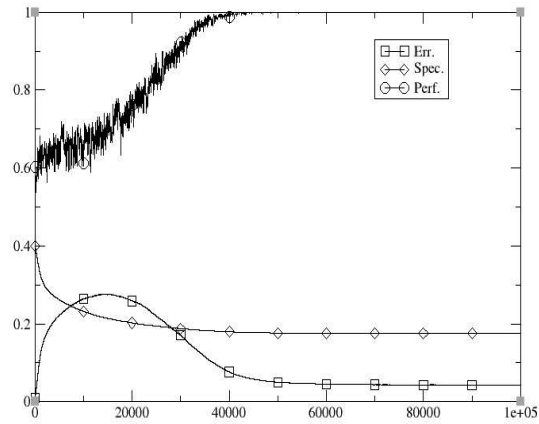


Figure 8: Ensemble on the 37mux problem.

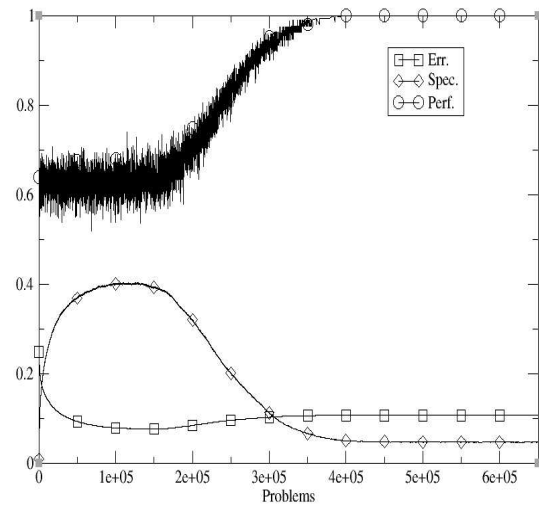


Figure 9: Ensemble on the 70mux problem.

7 Conclusions

This paper has presented an investigation into exploiting the population-based nature of Learning Classifier Systems for their use within highly-parallel data mining systems. We are particularly interested in the use of the data parallel ensemble machine approach with extremely large data sets, e.g., Terabytes, since it allows a divide-and-conquer approach to data manipulation (distributed storage). Results indicate that inclusion of a rule migration mechanism inspired by Parallel Genetic Algorithms can be an effective way to improve learning speed in LCS-EM. A significant reduction in the testing

time taken to achieve optimality was seen here, with better than linear reductions on the harder tasks. That is, despite the niche-based rule discovery mechanism of such accuracy-based LCS, the LCS-EM appears to benefit from the imposed population structure (mating restrictions) as do traditional evolutionary algorithms, as discussed in Section 5.

The system as a whole received ten times as many evaluations as the traditional single LCS system of course, but under a parallel implementation this is not the main concern; with very large data sets, the actual data processing time to build effective models is the critical factor even if the overall speed-up is not linear (or better). We are currently incorporating the approach into a super-computer data mining resource.

Future work, drawing on the existing ensemble machine literature (e.g., [Kuncheva & Whitaker, 2003]), will consider the effects of rule-base diversity both by extension to the model presented and empirically.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council, U.K. under grants GR/T18455 and GR/T18479.

Bibliography

Bagnall, A. & Smith, G. (1999) Using an Adaptive Agent to Bid in a Simplified Model of the UK Market in Electricity.

In W. Banzhaf et al. (eds) *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, pp774.

Bernadó, E., Llorà, X. & Garrell, J. (2002) XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks. In P-L. Lanzi, W. Stolzmann, S.W. Wilson (eds), *Advances in Learning Classifier Systems – IWLCS 2001*. Springer, pp115-133.

Bonelli, P. & Parodi, A. (1991) An Efficient Classifier System and its Experimental Comparison with two Representative Learning Methods on Three Medical Domains. In R. Belew & L. Booker (eds) *Proceedings of the 4th International Conference on Genetic Algorithms*. Morgan Kaufman, pp288-295.

Brown, G., Wyatt, J., Harris, R. & Yao, X. (2004) Diversity Creation Methods: A Survey and Categorisation. *Elsevier Science*

Bull, L. (1999) On using ZCS in a Simulated Continuous Double-Auction Market. In W. Banzhaf et al., (eds) *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, pp83-90.

Bull, L. (2002) On Accuracy-based Fitness. *Soft Computing* 6(3-4): 154-161.

- Bull, L. (2004)(ed) *Applications of Learning Classifier Systems*. Springer.
- Bull, L., Sha'Aban, J., Tomlinson, A., Addison, P. & Heydecker, B. (2004) Towards Distributed Adaptive Control for Road Traffic Junction Signals using Learning Classifier Systems. In L. Bull (ed) *Applications of Learning Classifier Systems*. Springer, pp276-299.
- Bull, L. (2005) Two Simple Learning Classifier Systems. In L. Bull & T. Kovacs (eds) *Foundations of Learning Classifier Systems*. Springer, pp63-90.
- Butz, M., Kovacs, T., Lanzi, P.L. & Wilson, S.W. (2004) Toward a Theory of Generalization and Learning in XCS. *IEEE Transactions on Evolutionary Computation* 8(1): 28.
- Cantu-Paz, E. (2000) *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer.
- Cantu-Paz, E. & Kamath, C. (2003) Inducing Oblique Decision Trees with Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 7(1): 54-68.
- Celis, S. & Musicant, D.R. (2003) Weka-Parallel. *Technical Report* <http://weka-parallel.sourceforge.net/>
- Cohon, J. Hegde, S., Martin, W. & Richards, D. (1987) Punctuated Equilibria: A Parallel Genetic Algorithm. In J.J. Grefenstette (ed) *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum, pp148-154.
- Dietterich, T. (2000) An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning* 40(2): 139-158.
- Dorigo, M. & Colombetti, M. (1997) *Robot Shaping*. MIT Press.
- Folino, G., Pizzuti, C. & Spezzano, G. (2000) Genetic Programming and Simulated Annealing: A Hybrid Method to Evolve Decision Trees. In *Proceedings of the Third European Conference on Genetic Programming, EuroGP2000*. Springer, pp. 294-303.
- Freitas, A. & Lavington, S.H. (1998) *Mining Very Large Databases with Parallel Processing*. Kluwer.
- Greenver, A. (2000) The use of a Learning Classifier System JXCS. In P. van der Putten & M. van Someren (eds) *CoIL Challenge 2000: The Insurance Company Case*. Technical Report 2000-09, Leiden Institute of Advanced Computer Science.
- Hercog, L. & Fogarty, T. (2002) Coevolutionary Classifier Systems for Multi-Agent Simulation. In *Proceedings of IEEE Congress on Evolutionary Computation*. IEEE Press, pp1789-1803.
- Ho, T., Hull, J.J. & Srikari, S.N. (1994) Decision Combination in Multiple Classifier Systems. *IEEE Trans on PAMI* 16 (1): 66-75.
- Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

- Holland, J.H. (1976) Adaptation. In R. Rosen & F.M. Snell (eds) *Progress in Theoretical Biology 4*. Plenum.
- Holland, J.H. & Miller, J. (1991) Artificially Adaptive Agents in Economic Theory. *American Economic Review* 81(2): 365-370.
- James, M. (1985) *Classification Algorithms*. Wiley.
- Kovacs, T. (2000) Strength or Accuracy? A Comparison of Two Approaches to Fitness Calculation in Learning Classifier Systems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp194-208.
- Kovacs, T. (2001) Toward a Theory of Strong Overgeneral Classifiers. In W. Martin & W. Spears (eds) *Foundations of Genetic Algorithms 6*. Morgan Kaufmann, pp165-184.
- Kunchevar, L. & Whitaker, C. (2003) Measures of Diversity in Classifier Ensembles. *Machine Learning* 51: 181-207.
- Llora, X. (2002) *Genetic Based Machine Learning using Fine-grained Parallelism for Data Mining*. PhD Thesis, Ramon Llull Univ.
- Manderick, B. & Spiessens, P. (1989) Fine-Grained Parallel Genetic Algorithms. In J.D. Schaffer (ed) *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, pp428-433.
- Marimon, R., McGrattan, E. & Sargent, T. (1990) Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents. *Economic Dynamics and Control* 14: 329-373.
- Neri, F. & Giordana, A. (1995) A Parallel Genetic Algorithm for Concept Learning. In T. Baeck (ed) *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp436-443.
- Palmer, R., Arthur, W., Holland, J.H., LeBaron, B. & Tayler, P. (1994) Artificial Economic Life: A Simple Model of a Stockmarket. *Physica D* 75: 264-274.
- Quinlan, J.R. (1993) *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
- Seredynski, F., Cichosz, P. & Klebus, G. (1995) Learning Classifier Systems in Multi-Agent Environments. In *Proceedings of the First IEE/IEEE Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*. IEE, pp287-292.
- Shafer, J., Agrawal, R. & Mehta, M. (1996) SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proceedings of the 22nd Very Large Data Bases Conference*. Morgan Kauffman, pp78-84.
- Smith, S.F. (1980) A Learning System Based on Genetic Adaptive Algorithms. PhD Diss., Univ. of Pittsburgh.
- Wilson, S.W. (1995) Classifier Fitness Based on Accuracy. *Evolutionary Computing* 3: 149-175
- Wilson, S.W. (2001) Mining Oblique Data with XCS. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (eds) *Advances in Learning Classifier Systems - IWLCS-2000*. Springer, pp158-176.

Zaki, M.J. & Ho, C-T. (2000) *Large-Scale Parallel Data Mining*. Springer.