Learning Classifier System Ensembles With Rule-Sharing

Larry Bull, Matthew Studley, Anthony Bagnall, and Ian Whittley

***Abstract*—This paper presents an investigation into exploiting the population-based nature of learning classifier systems (LCSs) for their use within highly parallel systems. In particular, the use of simple payoff and accuracy-based LCSs 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 equivalent single systems. Presentation of a mechanism which exploits the underlying niche-based generalization mechanism of accuracy-based systems is then shown to further improve their performance, particularly, as task complexity increases. This is not found to be the case for payoff-based systems. Finally, considerably better than linear speedup is demonstrated with the accuracy-based systems on a version of the well-known Boolean logic benchmark task used throughout.**

***Index Terms*—Data mining, genetic algorithms (GAs), parallel systems, reinforcement learning.**

I. Introduction

It is now widely recognized 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 [40]). We are currently producing a supercomputing data mining resource which utilizes 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 supercomputing.

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 $\langle \text{condition} \rangle$ THEN $\langle \text{class} \rangle$. In these systems, known as learning classifier systems (LCSs) [23], either each candidate solution is a complete set of rules [37] or the EA manipulates individual rules within a single rule-base/population (as in [23]). 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 [41] and health informatics [3]. Bernado *et al.* [2] have shown how the most current versions of the two LCS formalisms—GALE [29] and extended classifier system (XCS) [38], respectively—are extremely competitive with a number of well-known machine learning techniques (see also [7] for examples). XCS, in particular, has demonstrated excellent performance on a number of data mining tasks, see [19] and [39] for early examples. XCS builds a complete map of the given problem surface. That is, a genetic algorithm (GA) [22] 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., [10]), with similar findings for a version of Holland's algorithm (e.g., [9]). Importantly, this feature allows the user to examine how the LCS has achieved its performance. In this paper, we use both a simplified version of Holland's algorithm and a simplified version of XCS.

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) [34], artificial neural networks, instance-based algorithms, statistical approaches (e.g., [25]), and so on. 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, SPRINT [36] and Weka-parallel [13] (overviews can be found in [18] and [40]). 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., [15]). Versions of this EN approach, particularly, nonoverlapping initial training data systems, would also appear to be ideally suited for use on a supercomputing 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.

Manuscript received April 19, 2006; revised July 11, 2006. This work was supported in part by the Engineering and Physical Sciences Research Council, U.K., under Grant GR/T18455 and Grant GR/T18479.

L. Bull and M. Studley are with the School of Computer Science, University of the West of England, Bristol, BS16 1QY, U.K. (e-mail: larry.bull@uwe.ac.uk; matthew2.studley@uwe.ac.uk).

A. Bagnall and I. Whittley are with the School of Computer Sciences, University of East Anglia, Norwich NR4 7TJ, U.K. (e-mail: ajb@cmp.uea.ac.uk; imw@cmp.uea.ac.uk).

Digital Object Identifier 10.1109/TEVC.2006.885163

II. Two Simple LCS

A. YCS

In this paper, we use a version of the simple accuracy-based LCS termed YCS [9], 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

$$\varepsilon_j \leftarrow \varepsilon_j + \beta(|P - p_j| - \varepsilon_j) \tag{1}$$

$$\sigma_j \leftarrow \sigma_j + \beta(|[A]| - \sigma_j) \tag{2}$$

$$p_j \leftarrow p_j + \beta(P - p_j). \tag{3}$$

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

$$f_j = \left(1 \Big/ \left(\varepsilon_j^v + 1\right)\right). \tag{4}$$

Here, the exponent v enables control of the fitness pressure within the system by facilitating tuneable fitness separation under fitness proportionate selection (see [9] 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 [9] and [10] for discussions). The mechanism uses XCS's time-based approach under which each rule maintains a timestamp 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 timestamp 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, while remaining amenable to close modeling [9]: "[Each 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" [38], 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].

B. Minimal Classifier System (MCS)

In this paper, we use the simple payoff-based LCS termed MCS [9], which is a derivative of zeroth-level classifier system (ZCS) [38]. MCS is a minimal system without internal memory, where 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 fitness scalar (f) and the initial random population have the parameter initialized to 5 (f_0) here.

The matching procedure and formation of action sets is as described for YCS. Again, a variety of action selection schemes are possible but a version of XCS's explore/exploit 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 total payoff is chosen deterministically.

Although the use of fitness sharing for externally received payoff has been suggested before [22], it was not until Wilson introduced the action set-based scheme in ZCS that simple but effective fitness sharing in LCS became possible [9]. MCS uses the fitness sharing mechanism of ZCS, i.e., within action sets. The simplest case of immediate reward (payoff P) is again considered, and hence reinforcement consists of updating the fitness of each member of the current [A] using the Widrow–Hoff delta rule with learning rate β

$$f_j \leftarrow f_j + \beta((P/|[\mathrm{A}]|) - f_j). \tag{5}$$

MCS employs two discovery mechanisms, a panmictic GA and a covering operator. On each time-step there is a probability g of GA invocation. When called, the GA uses roulette wheel selection to determine two parent rules based on their fitness (5). Offspring are produced via mutation (probability μ turned into a wildcard at rate $p_\#$) and crossover (single point with probabilityχ), inheriting the parents' fitness values or their average if crossover is invoked. Replacement of existing members of the rulebase is inversely proportional to fitness, i.e., $(1/f_j + 1)$, using roulette wheel selection. If no rules match on a given time step, then a covering operator is used as described for YCS. Again, the GA is not invoked on exploit trials.

In this paper, to aid the generalization process, the GA is altered to include a heuristic which lowers the fitness of breeding rules and their offspring, in this case, all four have fitnesses assigned to f_0 (see [9] for discussions).

There are a few differences between MCS and ZCS. In particular, there is no fitness tax on the members of a matchset not forming the current [A] and rules do not donate 1/2 of their fitness to their offspring in MCS. Also, cover is not fired if the fitness of a matchset is a defined fraction below the population mean in MCS.

III. 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.

Fig. 1(a) shows the performance of YCS on the 20-bit multiplexer problem with $N = 2000, p_{\#} = 0.6, \mu = 0.04, v = 10, \chi = 0.5, \theta_{\mathrm{GA}} = 25$, and $\beta = 0.2$ (from [9]). After [38], performance from exploit trials is only 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 trails, matching that of XCS (e.g., [10]), with the average error dropping to roughly 10% of the payoff range. Fig. 1 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. Fig. 1(b) shows how MCS gives the same performance, here using the same parameters as YCS except $\mu = 0.01$.

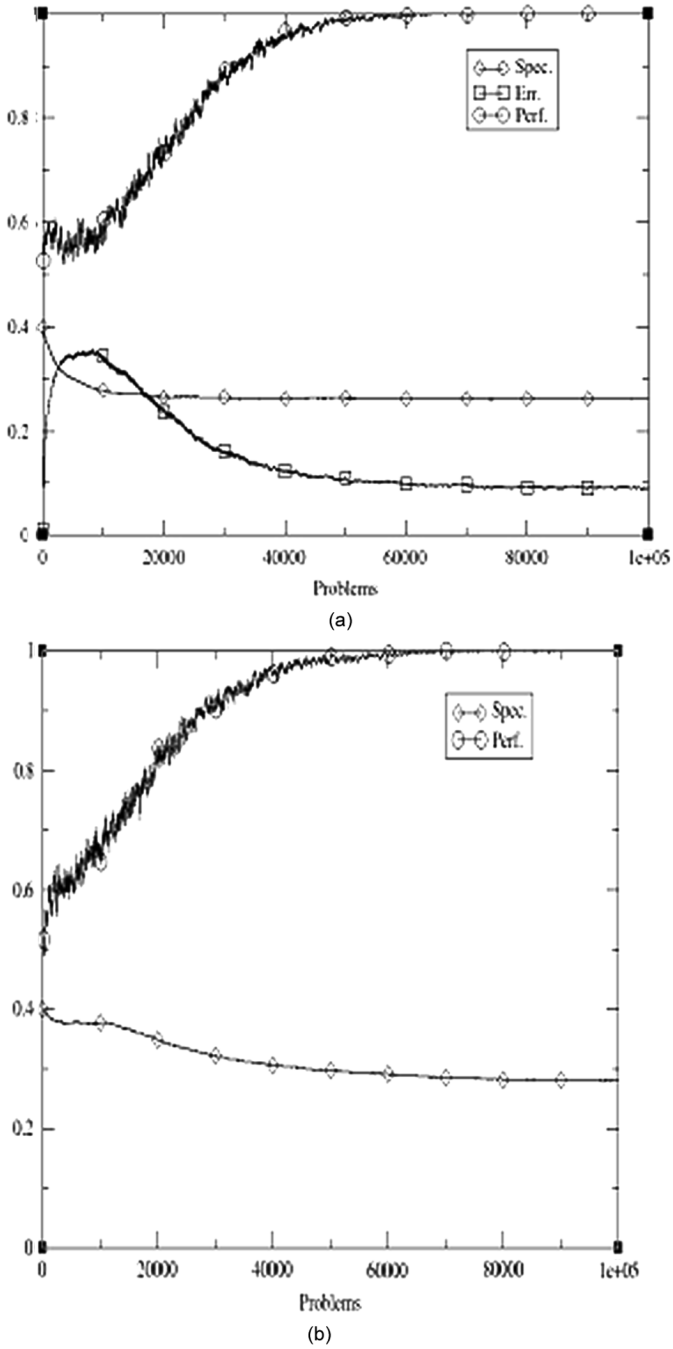

Fig. 1. Performance of (a) YCS and (b) MCS on the 20 mux problem.

IV. Ensembles of LCS

EMs (e.g., [21]) 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 utilizes the strength of the constituents and compensates for their individual weaknesses. This combination of algorithms is often constructed by measuring and maximizing the diversity (complementary characteristics) between the individuals, although its predictive power remains unclear (e.g., [4]). Perhaps most related to this work, Cantu-Paz and Kamath [12] used a simple EA to successfully design decision trees within an ensemble.

A number of investigators have examined the use of LCS in other multisystem environments. Early examples include Dorigo *et al.*'s (see [16] for an overview) use of multiple LCS within a hierarchical structure to control an autonomous robot and Seredynski *et al.*'s [35] examination of reward sharing in a simple iterated game. More recently, Hercog and Fogarty (e.g., [20]) used LCS to represent the agents of a simple game and Bull *et al.* (e.g., [8]) used LCS to control road-traffic junctions. There has also been some related work within the field of computational economics. After Holland and Miller [24], a number of researchers have used LCS to represent traders within artificial markets. Early examples include [31] and [33]. More recently, Bull (e.g., [5]) considered continuous double-auction markets and Bagnall and Smith (e.g., [1]) modeled the energy producers of the U.K. electricity market.

Fig. 2 shows the performance of an ensemble of ten YCS, each with the same parameters, as that shown in Fig. 1(a). 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 five or more LCS give the correct answer, the trial is considered to have been a success. As can be seen,

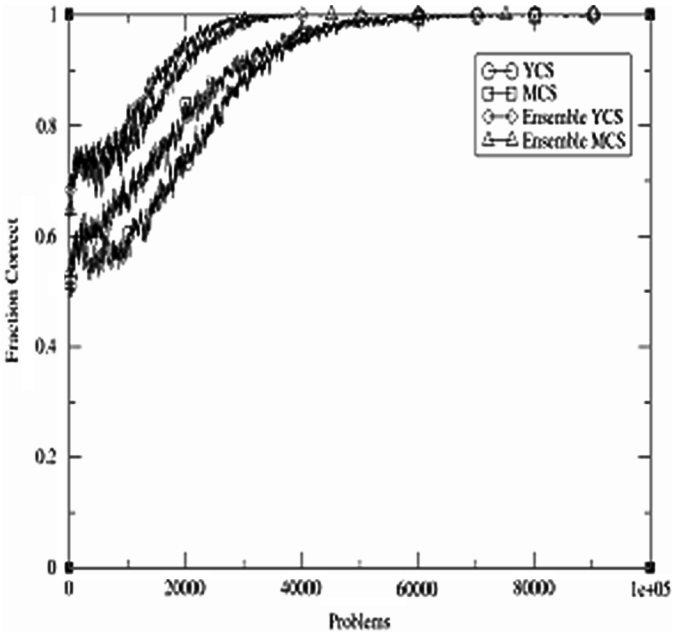

Fig. 2. Performance of LCS ensemble of ten systems versus single system on the 20 mux problem.

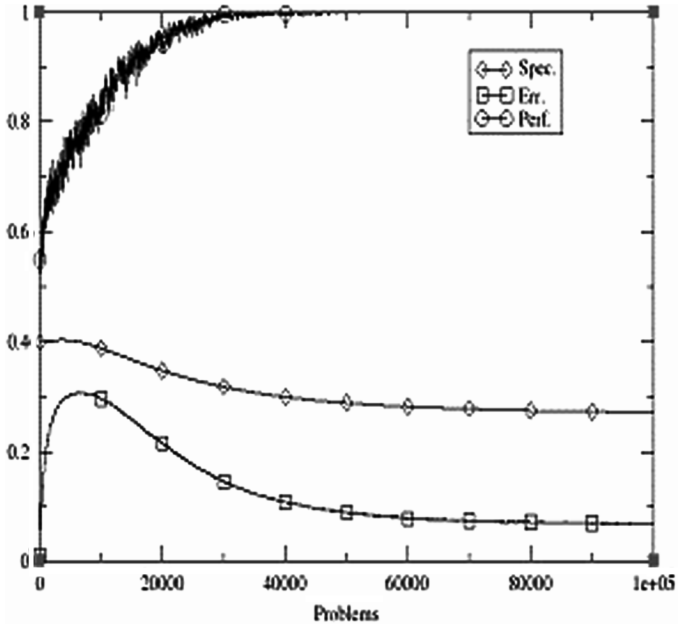

Fig. 3. Performance of YCS with $N = 20\,000$.

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 tenfold reduction is, of course, unlikely. A two-tailed T-test on the explore trial at which the succeeding 50 trials are correct shows, the ensemble is better than the former $(\mathrm{P} < 0.01)$. Fig. 2 also shows the performance of an ensemble of ten MCS, each with the same parameters as in Fig. 1(b) under the same scheme as the YCS ensemble. Again, a reduction in time taken to reach optimality is seen, this being statistically significantly better under the test described above than the single MCS $(\mathrm{P} < 0.01)$, and statistically insignificantly different from the YCS ensemble.

We note that the overall ensemble contains ten times as many rules as the traditional single system. Fig. 3 shows how using a rule-base of $N = 20\,000$ in a single YCS provides the same performance as the ensembles. However, results with the same rule resource in MCS give a significant decrease in performance (not shown). Given the process of fitness sharing within payoff-based LCS is population-wide [9], this result is perhaps not unsurprising and improved performance can be expected with different parameterization, although this has not been explored here. 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.

V. Rule Sharing in Ensembles

For nontrivial 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"—subpopulations in a hypercube—between which individuals are periodically shared (e.g., [14]), or a planar grid of processors—one candidate solution per node—across which good solutions diffuse through localized breeding (e.g., [30]). 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., [11]). Previous work on the evolution of data miners using a fine-grained EA include [17] and with the island approach [32].

Given that each LCS 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 supercomputer 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 in the given LCS. 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 in the given LCS.

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.

Fig. 4(a) shows the performance of the rule sharing ensemble of ten 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. The case with $\varphi = 0.001$ and $\phi = 1\%$ is shown. These two values are taken as "low" here. Similarly, it 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 both cases (and their intermediaries, not shown)—around 20 000 trials, a reduction in

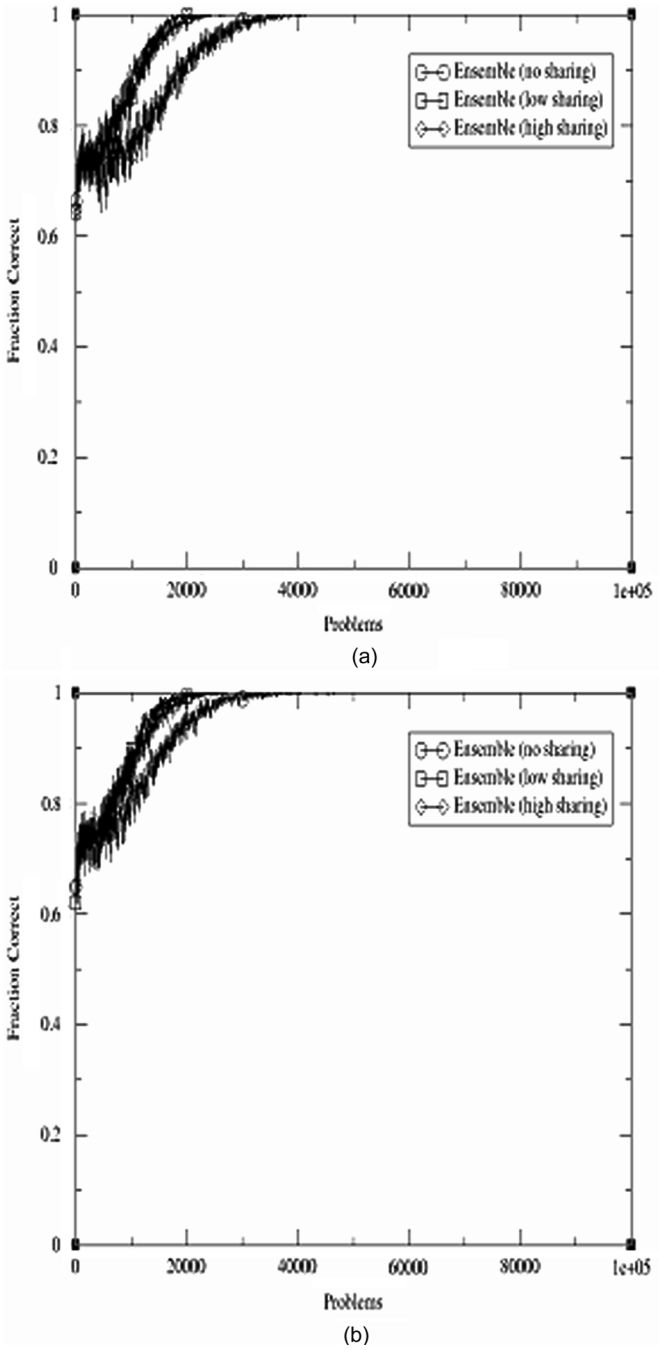

Fig. 4. Performance of (a) YCS ensembles and (b) MCS ensembles on the 20 mux problem with different sharing rate and fractions.

time of 2/3 over the single LCS. Thus, rule-sharing is beneficial ($\mathrm{P} < 0.05$ under the test described above) to the LCS ensemble machine. There is no statistically significant 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 (not shown). Conversely, empirical results with the island-model GA (e.g., [14]) suggest that one parameter should be high, with the other low for maximized benefit from migration events. Fig. 4(b) shows the same is also true for the equivalent MCS ensembles, with no statistically significant difference in performance compared with the YCS rule-sharing ensembles.

VI. Niche-Based Rule Sharing

As noted above, it is well-established (e.g., [10]) 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 timestamp 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 timestamp 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 rule-base of another member of the ensemble as described above. Again, this only occurs on explore trials.

Fig. 5 shows the performance of the YCS ensemble with $\theta_{\mathrm{SH}} = 10$. As can be seen, the time taken to reach optimal behavior is reduced to around 10 000 trials, this being statistically significantly better than the panmictic rule-sharing ensembles of Section V ($\mathrm{P} < 0.05$ under the test described above). Performance is shown relative to the original YCS ensemble without rule-sharing to indicate the performance increase now possible ($\mathrm{P} < 0.01$). Experiments with the panmictic scheme indicate that much higher levels of migration, e.g., $\varphi = 0.1$, do not give faster performance than before (not shown). With a larger value of θ_{SH} (e.g., $\theta_{\mathrm{SH}} = 100$), no statistically significantly different performance is obtained from this mechanism (not shown). 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 YCS ensemble. This scheme would also result in less data being passed interprocessor than under the panmictic scheme.

Results from using the same niche-based scheme within MCS indicate significantly worse performance. Again, as noted above, given the global processes of fitness sharing, this is not unexpected; the potentially disruptive effects of niche-based reproduction in payoff-based LCS have been discussed elsewhere [9].

Finally, we have examined the performance of the YCS ensemble using this migration scheme on two larger versions of the multiplexer task. Fig. 6 shows the performance of the same system as in Fig. 5 but with $N = 5000$ on the 37-bit

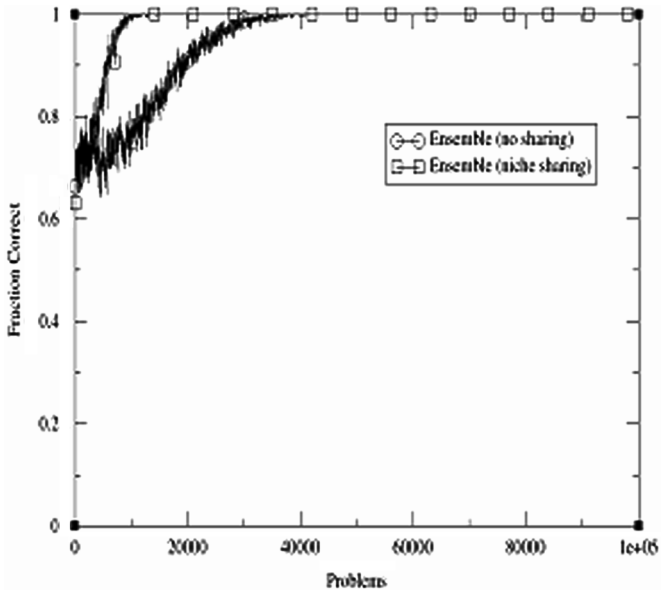

Fig. 5. YCS ensemble with $\theta_{\mathrm{SH}} = 10$ versus YCS ensemble without sharing.

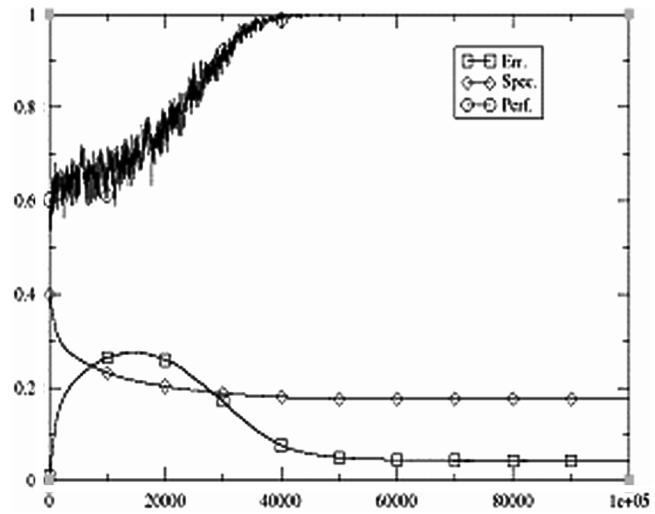

Fig. 6. YCS ensemble on the 37 mux problem.

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 and MCS with the same parameters (not shown, see [9]—also same for XCS e.g., [10]). Fig. 7 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.* [10] 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 speedup is better than linear here*.

VII. Conclusion

This paper has presented an investigation into exploiting the population-based nature of LCSs for their use within highly parallel data mining systems. We are particularly interested in the use of the data parallel EN approach with extremely large data sets, e.g., Terabytes, since it allows a divide-and-conquer

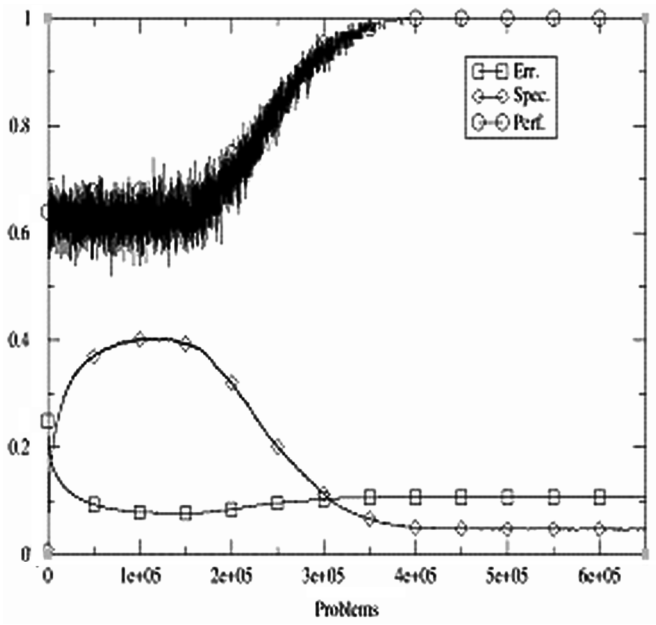

Fig. 7. YCS ensemble on the 70 mux problem.

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. That is, the LCS-EM appears to benefit from the imposed population structure (mating restrictions) as do traditional evolutionary algorithms, as discussed in Section V. Further, better than linear reductions in learning speed on the harder tasks using the accuracy-based LCS and a niche-based sharing mechanism were shown. It may be possible to match this performance with payoff-based LCS under a different rule-sharing mechanism but such a scheme has not been identified at this time, suggesting that such accuracy-based LCS are generally better suited to use within LCS-EM for such tasks.

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 speedup is not linear (or better). We are currently incorporating the approach into a supercomputer data mining resource.

Future work, drawing on the existing EN literature (e.g., [28]), will further consider the effects of rule-base diversity.

References

[1] A. Bagnall and G. Smith, "Using an adaptive agent to bid in a simplified model of the UK market in electricity," in *Proc. Genetic Evol. Comput. Conf.*, W. Banzhaf, Ed., Morgan Kaufmann, 1999, p. 74.

[2] E. Bernadó, X. Llorà, and J. Garrell, , P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., "XCS and GALE: A comparative study of two learning classifier systems with six other learning algorithms on classification tasks," in *Advances in Learning Classifier Systems—IWLCS 2001*. Berlin, Germany: Springer-Verlag, 2002, pp. 115–133.

[3] P. Bonelli and A. Parodi, R. Belew and L. Booker, Eds., "An efficient classifier system and its experimental comparison with two representative learning methods on three medical domains," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 288–295.

[4] G. Brown, J. Wyatt, R. Harris, and X. Yao, *Diversity Creation Methods: A Survey and Categorisation*. Amsterdam, The Netherlands: Elsevier Science, 2004.

[5] L. Bull, "On using ZCS in a simulated continuous double-auction market," in *Proc. Genetic Evol. Comput. Conf.*, W. Banzhaf, Ed., 1999, pp. 83–90.
[6] ——, "On accuracy-based fitness," *Soft Computing*, vol. 6, no. 3–4, pp. 154–161, 2002.
[7] ——, *Applications of Learning Classifier Systems*. Berlin, Germany: Springer-Verlag, 2004.
[8] L. Bull, J. Sha'Aban, A. Tomlinson, P. Addison, and B. Heydecker, "Towards distributed adaptive control for road traffic junction signals using learning classifier systems," in *Applications of Learning Classifier Systems*, L. Bull, Ed. Berlin, Germany: Springer-Verlag, 2004, pp. 276–299.
[9] L. Bull, "Two simple learning classifier systems," in *Foundations of Learning Classifier Systems*, L. Bull and T. Kovacs, Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 63–90.
[10] M. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, p. 28, Feb. 2004.
[11] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA: Kluwer, 2000.
[12] E. Cantu-Paz and C. Kamath, "Inducing oblique decision trees with evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 7, no. 1, pp. 54–68, Feb. 2003.
[13] S. Celis and D. R. Musicant, Weka-Parallel Tech. Rep., 2003. [Online]. Available: http://weka-parallel.sourceforge.net/
[14] J. Cohoon, S. Hegde, W. Martin, and D. Richards, "Punctuated equilibria: A parallel genetic algorithm," in *Proc. 2nd Int. Conf. Genetic Algorithms*, J. J. Grefenstette, Ed., 1987, pp. 148–154.
[15] T. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization," *Mach. Learn.*, vol. 40, no. 2, pp. 139–158, 2000.
[16] M. Dorigo and M. Colombetti, *Robot Shaping*. Cambridge, MA: MIT Press, 1997.
[17] G. Folino, C. Pizzuti, and G. Spezzano, "Genetic programming and simulated annealing: A hybrid method to evolve decision trees," in *Proc. 3rd Eur. Con. Genetic Program.*, 2000, pp. 294–303.
[18] A. Freitas and S. H. Lavington, *Mining Very Large Databases with Parallel Processing*. Norwell, MA: Kluwer, 1998.
[19] A. Greenver, "The use of a Learning Classifier System JXCS," CoIL Challenge 2000: The Insurance Company Case Leiden Institute of Advanced Computer Science, Tech. Rep. 2000-09, 2000, In P. van der Putten and M. van Someren, Eds..
[20] L. Hercog and T. Fogarty, "Coevolutionary classifier systems for multi-agent simulation," in *Proc. IEEE Congr. Evol. Comput.*, 2002, pp. 1789–1803.
[21] T. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. on PAMI 16*, no. 1, pp. 66–75, 1994.
[22] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.
[23] J. H. Holland, *Progress in Theoretical Biology 4*, R. Rosen and F. M. Snell, Eds. New York: Plenum, 1976, Adaptation.
[24] J. H. Holland and J. Miller, "Artificially adaptive agents in economic theory," *Amer. Econ. Rev.*, vol. 81, no. 2, pp. 365–370, 1991.
[25] M. James, *Classification Algorithms*. New York: Wiley, 1985.
[26] T. Kovacs, "Strength or accuracy? A comparison of two approaches to fitness calculation in learning classifier systems," in *Learning Classifier Systems: From Foundations to Applications*, P.-L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 194–208.
[27] T. Kovacs, "Toward a theory of strong overgeneral classifiers," in *Foundations of Genetic Algorithms 6*, W. Martin and W. Spears, Eds. San Mateo, CA: Morgan Kaufmann, 2001, pp. 165–184.
[28] L. Kunchevar and C. Whitaker, "Measures of diversity in classifier ensembles," *Mach. Learn.*, vol. 51, pp. 181–207, 2003.
[29] X. Llora, "Genetic based machine learning using fine-grained parallelism for data mining," Ph.D. dissertation, Ramon Llull Univ., Barcelona, Spain, 2002.
[30] B. Manderick and P. Spiessens, "Fine-grained parallel genetic algorithms," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. D. Schaffer, Ed., 1989, pp. 428–433.
[31] R. Marimon, E. McGrattan, and T. Sargent, "Money as a medium of exchange in an economy with artificially intelligent agents," *Economic Dynamics and Control*, vol. 14, pp. 329–373, 1990.
[32] F. Neri and A. Giordana, "A parallel genetic algorithm for concept learning," in *Proc. 6th Int. Conf. Genetic Algorithms*, T. Baeck, Ed., 1995, pp. 436–443.
[33] R. Palmer, W. Arthur, J. H. Holland, B. LeBaron, and P. Tayler, "Artificial economic life: A simple model of a stockmarket," *Physica D*, vol. 75, pp. 264–274, 1994.
[34] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
[35] F. Seredynski, P. Cichosz, and G. Klebus, "Learning classifier systems in multi-agent environments," in *Proc. 1st IEE/IEEE Conf. Genetic Algorithms in Eng. Syst.: Innovations Appl.*, 1995, pp. 287–292.
[36] J. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A scalable parallel classifier for data mining," in *Proc. 22nd Very Large Data Bases Conf.*, 1996, pp. 78–84.
[37] S. F. Smith, "A learning system based on genetic adaptive algorithms," Ph.D. dissertation, Univ. Pittsburgh, Pittsburgh, PA, 1980.
[38] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, pp. 149–175, 1995.
[39] S. W. Wilson, "Mining Oblique Data with XCS," in *Proc. Advances in Learning Classifier Syst.*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., 2000, pp. 158–176.
[40] M. J. Zaki and C.-T. Ho, *Large-Scale Parallel Data Mining*. Berlin, Germany: Springer-Verlag, 2002.
[41] G. Robertson and R. Riolo, "A tale of two classifier systems," *Mach. Learn.*, vol. 3, pp. 139–159, 1988.

Larry Bull received the B.Sc. (Hons.) degree in computing for real-time systems and the Ph.D. degree from the University of the West of England (UWE), Bristol, U.K., in 1992 and 1995, respectively.

He is Professor of Artificial Intelligence within the Faculty of Computing, Engineering and Mathematical Sciences (CEMS), UWE. His research interests include evolutionary computing, reinforcement learning, neural computing, and biology-inspired artificial systems in general.

Matthew Studley received the B.Sc. degree in biological science from Sussex University, Brighton,, U.K., in 1990, the M.Sc. degree in parallel computing systems and the Ph.D. degree from the University of the West of England (UWE), Bristol, U.K, in 1996 and 2006, respectively. His Ph.D. research investigated the use of learning classifier systems to control physical robots in problems with more than one objective.

He has worked in the telecommunications, e-commerce, investment and finance sectors in many European countries, the U.S., and Australia, and is currently at UWE.

Anthony Bagnall received the B.Sc. degree from the University of Hertfordshire, Herts, U.K., in 1994, the M.Res. degree in computer science and the Ph.D. degree in computer science both from the University of East Anglia (UEA), Norwich, U.K., in 1996 in 2000, respectively.

He is a Senior Lecturer in Computer Science at the University of East Anglia. His research interests mainly lie in the fields of autonomous adaptive agents and data mining, with particular focus on learning classifier systems and time series data mining.

Ian Whittley received a First Class Honors degree in computer science from the University of East Anglia (UEA), Norwich, U.K., in 1996. This was followed by a period of research into the use of metaheuristics in optimization for which he received the Ph.D. degree from UEA, in 2001. Two parameter free local search algorithms developed as part of this work have been successfully applied to a range of academic and real-world problems.

He has worked as a Senior Research Associate at UEA for many years on projects that include the automated assessment of exams, data mining, and optimization.