LEARNING CLASSIFIER SYSTEMS

Larry Bull, chair

Lookahead and Latent Learning in ZCS

Larry Bull
Faculty of Computing, Engineering & Mathematical Sciences
University of the West of England
Bristol BS16 1QY
larry.bull@uwe.ac.uk

Abstract

Learning Classifier Systems use reinforcement learning, evolutionary computing and/or heuristics to develop adaptive systems. This paper extends the ZCS Learning Classifier System to improve its internal modelling capabilities. Initially, results are presented which show performance in a traditional reinforcement learning task incorporating lookahead within the rule structure. Then a mechanism for effective learning without external reward is examined which enables the simple learning system to build a full map of the task. That is, ZCS is shown to learn under a latent learning scenario using the lookahead scheme. Its ability to form maps in reinforcement learning tasks is then considered.

1 INTRODUCTION

Traditional Learning Classifier Systems (LCS) [Holland 1976] use genetic algorithms (GA) [Holland 1975] and the bucket brigade algorithm [Holland 1986] to produce an interacting ecology of rules for a given task. Holland et al. [1986] proposed a number of mechanisms by which LCS could potentially realise many complex inductive processes. However, the basic architecture was difficult to use and understand. Wilson [1994] presented ZCS which "keeps much of Holland's original framework but simplifies it to increase understandability and performance" [ibid.]. Bull and Hurst [2002] have recently shown that, despite its relative simplicity, ZCS is able to perform optimally through its use of fitness sharing. That is, ZCS was shown to perform as well, with appropriate parameters, as the more complex XCS [Wilson 1995] on a number of tasks. The significant difference between the two systems being XCS's ability to build a complete, maximally general map of the given problem.

In this paper the basic ZCS architecture is extended to include mechanisms by which cognitive capabilities, along the lines of those envisaged by Holland et al. [1986], can emerge; the use of predictive modelling within ZCS is considered through an alteration to the rule structure. Using a maze task loosely based on that of early animal behaviour experiments, it is found that ZCS can learn effectively when reward is dependent upon the ability to accurately predict the next environment state/sensory input. This ZCS with lookahead is then extended to work under latent learning. That is, an approach is presented which allows ZCS to build a full map of its task without external reinforcement. This result is then suggested as significant for traditional payoff-based LCS when the aforementioned difference to XCS is considered.

The paper is arranged as follows: the next section briefly describes ZCS. Section 3 considers the use of lookahead in general and presents results from its use within ZCS. In Section 4 the use of latent learning with the predictive form of ZCS is presented. Finally, all findings are discussed.

2 ZCS

ZCS is a "Zeroth-level" Michigan-style Classifier System without internal memory, where the rule-base 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 which acts as an indication of the perceived utility of that rule within the system. This fitness of each rule is initialised to a predetermined value termed S_0.

Reinforcement in ZCS consists of redistributing fitness between subsequent "action sets", or the matched rules from the previous time step which asserted the chosen output or "action". A fixed fraction (β) of the fitness of each member of the action set ([A]) at each time-step is placed in a "common bucket". A record is kept of the previous action set $[A]_{-1}$ and if this is not empty then the members of this action set each receive an equal share of the contents of the current bucket, once this has been reduced by a pre-

determined discount factor (γ). If a reward is received from the environment then a fixed fraction (β) of this value is distributed evenly amongst the members of [A]. Finally, a tax (τ) is imposed on all matched rules that do not belong to [A] on each time-step in order to encourage exploitation of the stronger classifiers. Wilson notes that this is a change to the traditional LCS bucket-brigade algorithm [Holland 1986] since there is no concept of a rule 'bid', generalisation is not considered explicitly, sets of rules are updated and the pay-back is reduced by 1-γ on each step (see [Bull & O'Hara 2001] for related discussions).

ZCS employs two discovery mechanisms, a panmictic GA and a covering operator. On each time-step there is a probability p of GA invocation. When called, the GA uses roulette wheel selection to determine two parent rules based on fitness. Two offspring are produced via mutation (probability μ, probability of inserting a wildcard $p_{\#}$) and crossover (single point with probability χ). The parents then donate half of their fitnesses to their offspring who replace existing members of the rule-base. The deleted rules are chosen using roulette wheel selection based on the reciprocal of rule fitness. If on some time-step, no rules match or all matched rules have a combined fitness of less than ϕ times the rule-base average, then a covering operator is invoked.

The default parameters presented for ZCS, and unless otherwise stated for this paper, are: $N = 400$, S_0=20, $\beta = 0.2$, $\gamma = 0.71$, $\tau = 0.1$, $\chi = 0.5$, $\mu = 0.002$, $p = 0.25$, $\phi = 0.5$, $p_{\#}$=0.33.

3 LOOKAHEAD

Holland [1990] presented a general framework for incorporating future state predictions into LCS, termed lookahead (after [Samuel 1959]). Lookahead allows a learning entity to construct an internal model of its environment, " ... a matter of implementing the rule 'IF the environment is in state S AND action A is taken THEN (the system expects) state S2 will occur'" [Holland 1990]. Under Holland's scheme tags, which (potentially) facilitate rule coupling under normal operation [Holland 1986], would be used rather than an explicit representation of the expected environmental state. A "virtual" bucket brigade algorithm would then pass payoff back through "cones" of likely future outcomes to influence the action selection process on any given step. Stolzmann [1998] has presented a heuristic-driven LCS, ACS, which uses the explicit next-state rule structure noted above to build anticipatory models of an environment. The accuracy of the rules' predictions are factored into their utility. Later work added a GA which used this measure for rule fitness resulting in improved performance [e.g. Butz et al. 2000]. LCS which use rule-linkage over succeeding timesteps [e.g. Tomlinson & Bull 1998] also implicitly build predictions of future states; the condition of a linked rule must represent the next environmental state after the action of its predecessor is taken.

3.1 THE APPROACH

In this paper, as in [Riolo 1991][Stolzmann 1998][Gerard & Sigaud 2001] and suggested in [Wilson 1995], an explicit representation of the expected next environmental state is used. That is, rules are of the general form:

<condition> : <action> : <anticipation>

Generalizations (#'s) are allowed in the condition and anticipation strings. Where #'s occur at the same loci in both, the corresponding environmental input symbol "passes through" such that it occurs in the anticipated description for that input. Similarly, defined loci in the condition appear when a # occurs in the corresponding locus of the anticipation. Each rule also maintains the usual fitness parameter as in ZCS.

One further mechanism is incorporated: the first N random rules of the rule-base have their anticipation created using cover (with #'s included as usual) in the first [A] of which they become a member. This goes some way to make " ... good use of the large flow of (non-performance) information supplied by the environment." [Holland 1990] and can be seen to create a supervised learning task during initialization (this is particularly significant in Section 5). Rules created under the cover operator also receive this treatment. In this way the GA explores the generalization space of the anticipations created by the simple heuristic (as opposed to [Stolzmann 1998]).

All other system functionality is as described in Section 2, except that members of a given [A] only receive bucket payments if they correctly predicted the next state. Hence, in effect, incorrect predictors are taxed at the learning rate. Predictions are not tested for external reward receiving rules.

3.2 THE TASK

The aim of this paper is to show that ZCS can be extended to exploit lookahead and latent learning to build a more comprehensive internal model of the task. The general motivation for such work with machine learning algorithms comes, in part, from experiments undertaken with rats by Tolman [e.g. see Mackintosh 1974], Seward [1949] and others. It was shown that rats appear able to construct internal models of simple mazes of the general form shown in Figure 1 so that, when later placed at the start (lowest cell), they would find the food via the shortest route. This will be returned to in Section 4, a goal-directed only version being examined here.

In this section the task is seen as the well-known animat

problem [Wilson 1987]. As such, ZCS is used to develop the controller of a simulated robot/animat which must traverse the maze in search of food. It is positioned randomly in one of the blank cells and can move into any one of the surrounding eight cells on each discrete time step, unless occupied by a tree. If the animat moves into the food cell the system receives a reward from the environment (1000), and the task is reset, i.e. food is replaced and the animat randomly relocated.

On each time step the animat receives a sensory message which describes the eight surrounding cells. The message is encoded as a 16-bit binary string with two bits representing each cardinal direction. A blank cell is represented by 00, food (F) by 11 and trees (T) by 10 (01 has no meaning). The message is ordered with the cell directly above the animat represented by the first bit-pair, and then proceeding clockwise around the animat.

The trial is repeated 10,000 times and a record is kept of a moving average (over the previous 50 trials) of how many

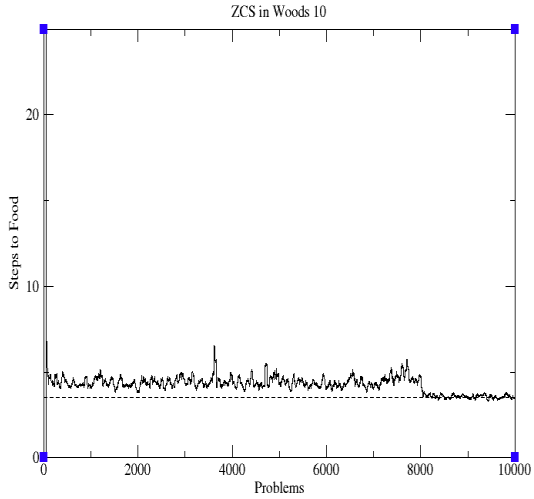
T	T	T	T	T	T	T
T	F	T	T	T		T
T		T	T	T		T
T						T
T	T	T		T	T	T
T	T	T		T	T	T
T	T	T	T	T	T	T

Figure 1: The Woods 10 environment.

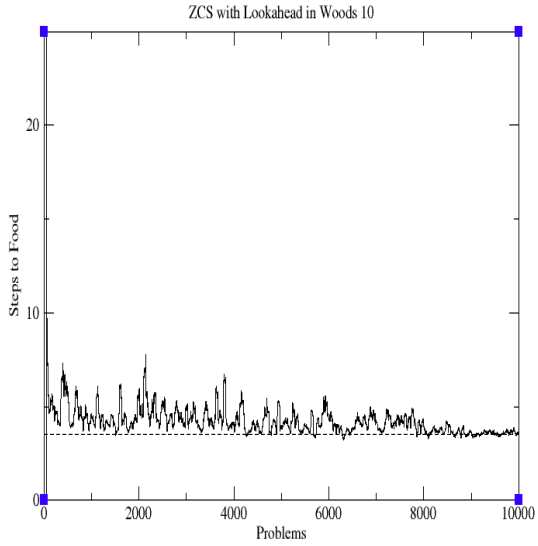
steps it takes for the animat to move into a food cell on each trial. Optimal performance is 3.5 steps to food. All results presented are the average of ten runs.

3.3 RESULTS

Figure 2(a) shows the performance of standard ZCS in Woods 10 with the same parameters as those in Section 2, except β=0.45. Optimal performance can be seen during the last 2000 trials where the GA was disabled and a deterministic action selection scheme used such that the action with the highest total fitness in [M] was always chosen (after [Bull & Hurst 2002]). Figure 2(b) shows the performance of ZCS using similar parameters but with the lookahead rule structure and scheme described above (γ=0.4, ρ=0.45). It can be seen that performance is equivalent and hence that ZCS is able to produce accurate next-state predictions. However, ZCS does not form a full state-action-anticipation map under reinforcement learning. The next section presents a mechanism by which such a map can be constructed under latent learning.



(a)



(b)

Figure 2: Performance of ZCS in Woods 10 and incorporating lookahead.

4 ZCSL: USING LATENT LEARNING WITH LOOKAHEAD

As noted above, one motivation for exploring the use of learning without external reinforcement comes from early experiments in animal behaviour. Typically, rats were allowed to run around a maze of the shape shown in Figure 1 where the food cell would be empty but a different colour to the rest of the maze. The rats would then be fed in the marked location. Finally, the rats were placed at the start location and their ability to take the shortest path, i.e. go left at the T-junction in Figure 1, recorded. It was found that rats could do this with around 90% efficiency. Those which were not given the prior experience without food were only 50% efficient, as expected.

Riolo [1991] extended a version of Holland's LCS to consider such learning without external reinforcement, using the same general rule form as that above and tags. The bucket brigade was then altered to consider the accuracy of rule's predictions of future states. Although Riolo did not incorporate rule discovery, he showed his CFCS2 could learn and exploit internal models to solve a version of the maze task described above. Both ACS [Stolzmann & Butz 2000] and the related YACS[Gerard & Sigaud 2001] have also been shown able to develop internal models under latent learning using heuristics.

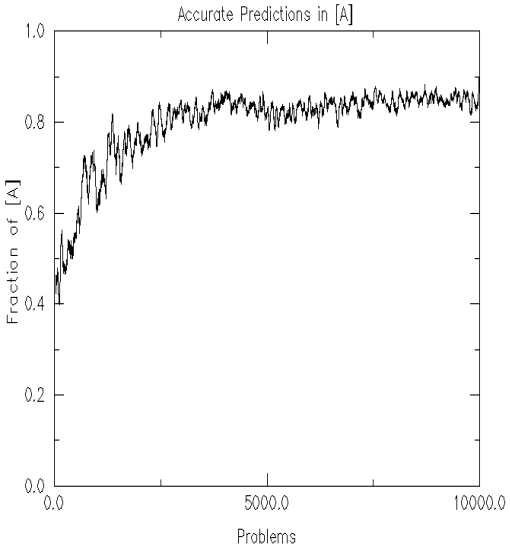
4.1 THE APPROACH

The *internal model building task can be cast as a single-step task*. In the simplest case, using lookahead under latent learning and the above rule structure, a single-step learning task is created whereby reward is given only if a rule predicts the expected outcome of taking its action under the condition matched. This is the approach used in ZCSL.

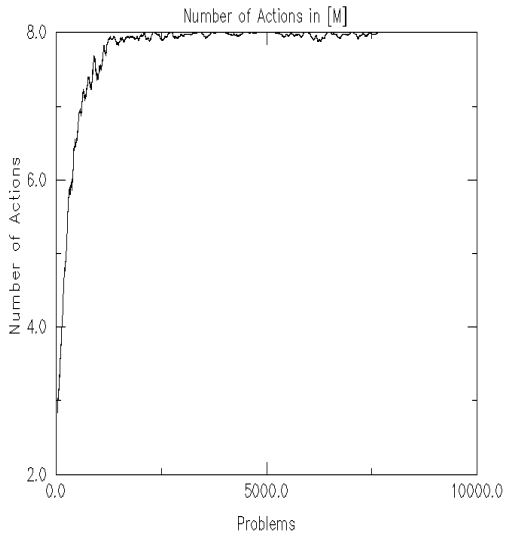
At the beginning of a trial, the animat is placed randomly in the maze. A matchset is formed and an action chosen at random. All rules which propose the chosen action form [A] and pay β of their fitness into the bucket as usual. All rules in [A] then construct their anticipated sensory input for the next state, i.e. pass-through is considered, and the action is taken. Each rule in [A] which correctly predicts the next state is rewarded with payoff 1000 divided by the number of correct rules in [A]; fitness sharing is used. Note that taxing the other members of the matchset is no longer appropriate as a full map of actions is required. This process is repeated for ten steps and then the animat is randomly replaced in the maze. It is important to note that although the animat can sense the food, it is not able to move onto that cell (but predictions are tested as if it had). All other operations are as before.

4.2 RESULTS

Figure 3 shows the behaviour of this form of ZCS in the Woods 10 maze. The parameters used were the same as those given in Section 2, except N=800, and the GA is again turned off for the last 2000 trials.



(a)



(b)

Figure 3: Showing the performance of ZCSL in the Woods 10 environment.

Figure 3(a) shows the fraction of rules of a given action set which correctly predicted the next environmental state. It can be seen that by around 5000 trials 80% (i.e. the majority) of the rules in each action set were accurate predictors. Figure 3(b) shows the number of actions represented in each matchset has risen to eight around the same time. Hence after 5000 trials (50,000 cycles) *ZCSL has constructed a full and accurate map of the maze*, assuming the most numerous anticipated next state of a given [A] is used. Note that ZCSL includes state-action pairs which lead to no change in stimulus without the explicit consideration of such circumstances. The original ACS did not develop rules for such cases, but was later modified [Stolzmann 2000] (see also YACS).

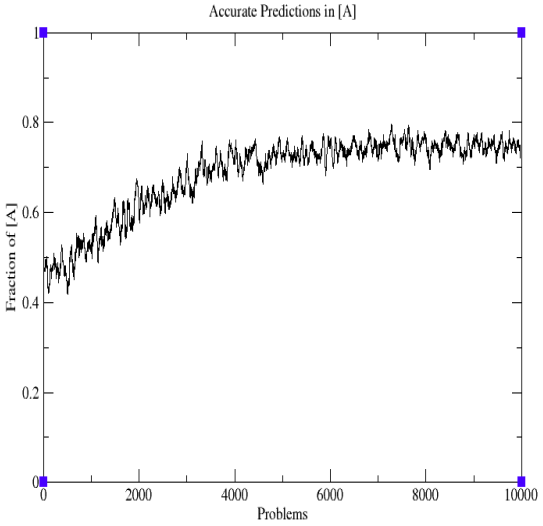
T	T	T	T	T	T	T	T	T
T						T	F	T
T			T		T	T		T
T		T						T
T				T	T			T
T		T		T				T
T		T						T
T						T		T
T	T	T	T	T	T	T	T	T

Figure 4: Maze 6.

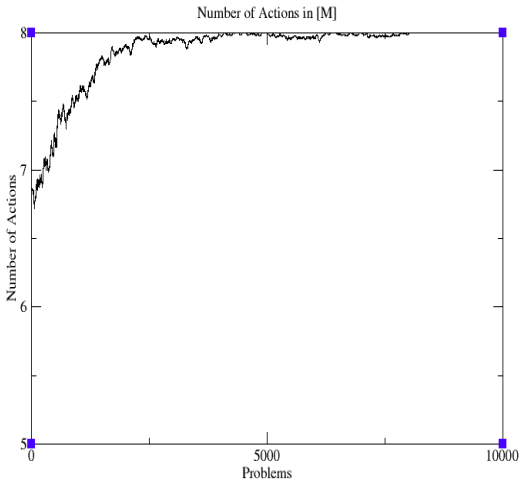
Therefore, the resulting LCS systems could be used to reproduce the rat experiments through any number of planning techniques, such as breadth-first search: from a given input the most numerous anticipation of each [A] can be considered to represent the next environmental input, and so on, until the goal state is seen. If this is done from the start location, firing the appropriate sequence of rules would give 100% efficiency at the task. That is, the ZCSL controllers have the ability to reproduce the general behaviour of the rats using a very simple LCS architecture. The following four rules show an example solution found in this way starting from the bottom cell (GA fitnesses not shown):

#0##1#101#1#1010 : N : #000101000#01#00
0000101000101000 : NW : 1#1#0#001010000#
####10100#0#101# : NW : #0###0#0#0#0#010
#1#01#00#010#0#0 : N : ##10#0#0##10#010

It can be noted that both generalization and pass-through are contained in the solutions although no explicit pressure for either exists within the simple system.



(a)



(b)

Figure 5: Showing the performance of ZCSL in the Maze 6 environment.

ZCSL has also been applied to the more challenging Maze 6 [Lanzi 1997] environment (Figure 4). The parameters used were the same as those above, except N=10,000. Again, the animat could not move onto the food cell. Figure 5(a) shows the fraction of rules of a given action set which correctly predicted the next environmental state. It can be seen that by around 5000 trials the majority of each action set (>75%) were accurate predictors. Figure 5(b) shows the number of actions represented in each matchset has again risen to eight around the same time. Hence, after 5000 trials ZCSL has constructed a full and accurate map of Maze 6.

5 DISCUSSION: XCS

As noted in the introduction, XCS attempts to build a full, non-overlapping, maximally general map of the problem space which can have advantages over traditional payoff-based LCS, such as ZCS, if *a posteri* explanatory power is required, as in data mining for example.

There seems no reason why the above modified ZCS system of Section 4 cannot also produce such predictive maps, particularly for single step tasks. That is, *if the anticipation is altered from being an expected environmental state to a numerical payoff value, a similar map can be formed*. I.e. rules are of the general form:

<condition> : <action> : <anticipated payoff>

This version has been explored using the well-known multiplexer task. These Boolean functions are defined for binary strings of length $l = k + 2^k$ under which the first 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.

All system functionality is as described in Section 4 except that the appropriate value (1000 or 0) is written on as the anticipation for a randomly created rule. Mutation causes a change in the value to another valid payoff level (1000 to 0, or vice versa here). A trial is two evaluations here.

Figure 6 shows the percentage of correct predictions in a given [A] and number of actions in [M] for the thirty seven bit multiplexer, using the same parameters as for Woods 10 except N=5000, β=0.8 and $p_{\#}$=0.8. It can be seen that, on average, correct predictors occur with highest numerosity and that both actions are present in each matchset after around 700,000 trials (1,400,000 evaluations). The evolved rules for the first two data lines from an example solution were as follows (GA fitnesses not shown):

000000################################ : 0 : 1000
000000############################0## : 1 : 0
000001############################0### : 0 : 0
000001################################ : 1 : 1000
00001#0############################### : 0 : 1000
00001#0##0##########0##########0## : 1 : 0
00001#1########0##################### : 0 : 0
00001#1############################### : 1 : 1000

It can again be noted that solutions are very general although no explicit pressure for this exists within the simple system. Results show that, the longer the system is left to run, the more general rules become.

Butz et al. [2001] have recently examined the behaviour of XCS on a number of multiplexer problems, also solving the thirty seven bit version. They note that a stronger distinction between rule accuracies was required in comparison to the smaller multiplexer tasks and a larger $p_{\#}$. ZCSL also appears to need a strong pressure toward accurate predictors through an increase in the learning rate since this is also essentially the tax rate for erroneous rules. A large $p_{\#}$ also proved important. Butz et al. did not manage to solve the equivalent seventy bit task and initial attempts with ZCSL have also proven unsuccessful. This remains open to future investigation.

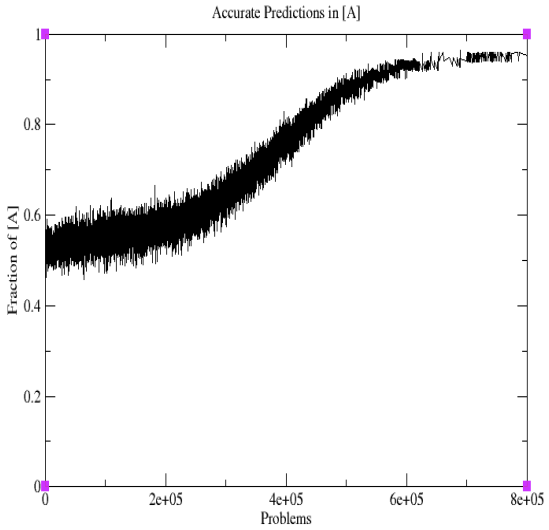
Hence this system uses a simple heuristic to promote accuracy in payoff predictions whilst the GA with fitness sharing apportions resources and encourages generalization. In contrast, XCS uses a four-step fitness update to promote accuracy in payoff predictions, an explicit replacement strategy to apportion (balance) resources and a triggered niche GA to encourage generalization. Whether the simple approach described here scales as well as XCS to tasks with more classes and prediction levels, noisy data, or can be used in multi-step tasks represents future work. The use of some of XCS's other features (subsumption, action set filling, etc.) may help.

6 CONCLUSIONS

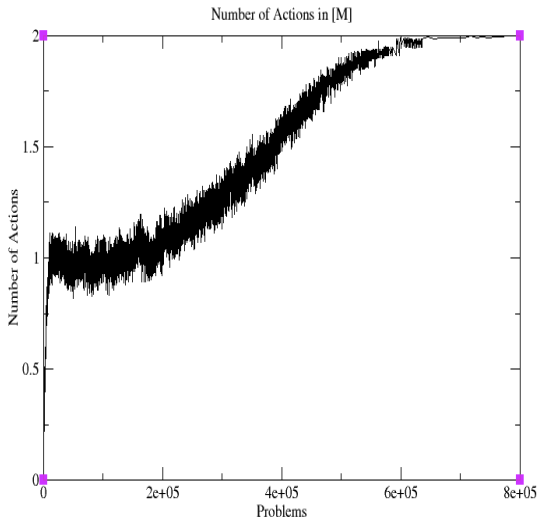
In this paper ZCS has been extended to incorporate lookahead and latent learning. Using a simple maze task, based on those used in early animal behaviour experiments, it has been shown that ZCS can build *partial* internal models under traditional goal-directed learning. The construction of a *full* internal environment model under latent learning with lookahead was then cast as a single-step reinforcement task and ZCSL was shown able to form accurate maps under fitness sharing. Future work will examine the inclusion of other mechanisms, such as an explicit unchanging component [e.g. Stolzmann 2000], to improve performance. Other schemes to encourage maximal generality within solutions will also be explored.

The use of the mechanisms within a more complex framework to exploit internal models during learning under reinforcement, after Sutton's Dyna [e.g. Sutton 1990] (see

also [Donnart & Meyer 1996][Stolzmann et al. 200]), is also under investigation.



(a)



(b)

Figure 6: Showing the performance of ZCSL on the 37 bit multiplexer problem.

ACKNOWLEDGEMENT

Thanks to the members of the Learning Classifier System Group at UWE for many useful discussions during this work**.**

REFERENCES

Bull, L. & Hurst, J. (2002) ZCS Redux. *Evolutionary Computation,* in press

Bull, L. & O'Hara, T. (2001) NCS: A Simple Neural Classifier System. *UWE Learning Classifier Systems Group Technical Report 01-005*. Available from http:// www.csm.uwe.ac.uk/lcsg.

Butz, M., Goldberg, D.E. & Stolzmann, W. (2000) Introducing a Genetic Generalization Pressure to the Anticipatory Classifier System: Part 1 - Theoretical Approach. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference - Gecco 2000*. Morgan Kaufmann, pp34-41.

Butz, M., Kovacs, T., Lanzi, P-L & Wilson, S.W. (2001) How XCS Evolves Accurate Classifiers. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference - Gecco 2001*. Morgan Kaufmann, pp927-934.

Donnart, J-Y. & Meyer, J-A. (1996) Spatial Exploration, Map Learning, and Self-Positioning with MonaLysa. In P. Maes, M. Mataric, J-A. Meyer, J. Pollack & S.W. Wilson (eds.) *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behaviour,* MIT Press, pp 204-213.

Gerard, P. & Sigaud, O. (2000) YACS: Combining Dynamic Programming with Generalization in Classifier Systems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop.* Springer, pp52-69.

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. (1986) Escaping Brittleness. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (eds) *Machine Learning: An Artificial Intelligence Approach, 2*. Morgan Kauffman, pp48-78

Holland, J.H. (1990) Concerning the Emergence of Tag-Mediated Lookahead in Classifier Systems. *Physica D* 42:188-201.

Holland, J.H., Holyoak, K.J., Nisbett, R.E. & Thagard, P.R. (1986) *Induction: Processes of Inference, Learning and Discovery*. MIT Press.

Lanzi, P-L. (1997) A Model of the Environment to Avoid Local Learning. Technical Report N.97.46 Dipartimento di Elettronica e Informazione, Politecnico di Milano.

Mackintosh, N.J. (1974) *The Psychology of Animal Learning.* Academic Press.

Riolo, R. (1991) Lookahead Planning and Latent Learning in a Classifier System. In J-A. Meyer & S.W. Wilson (eds.) *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behaviour*, MIT Press, pp316-326.

Samuel, A.L. (1959) Some Studies in Machine Learning using the Game of Checkers. *IBM Journal of Research and Development,* 3: 211-232.

Seward, J.P. (1949) An Experimental Analysis of Latent Learning. *Journal of Experimental Psychology* 39: 177-186.

Stolzmann, W. (1998) Anticipatory Classifier Systems. In J.R. Koza (ed) *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann, pp658-664.

Stolzmann, W. (2000) An Introduction to Anticipatory Classifier Systems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp175-194.

Stolzmann, W. & Butz, M. (2000) Latent Learning and Action Planning in Robots with Anticipatory Classifier Systems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp301-320.

Stolzmann, W., Butz, M., Hoffmann, J. & Goldberg, D.E. (2000) First Cognitive Capabilities in the Anticipatory Classifier System. In J-A. Meyer, A. Berthoz, D. Floreano, H. Roitblatt & S.W. Wilson (eds) *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behaviour,* MIT Press.

Sutton, R.S. (1990) Integrated Architectures for Learning, Planning and Reacting based on Approximating Dynamic Programming. *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, pp216-224.

Tomlinson, A. & Bull, L. (1998) A Corporate Classifier System. In A.E. Eiben, T. Bäck, M. Schoenauer & H-P. Schwefel (eds.) *Parallel Problem Solving from Nature - PPSN V*, Springer, pp. 550-559.

Wilson, S.W. (1987) Classifier Systems and the Animat Problem. *Machine Learning,* 2: 199-228.

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

Wilson, S.W. (1995) Classifier Fitness Based on Accuracy. *Evolutionary Computation* 3(2):149-177

Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems

Larry Bull
Faculty of Computing, Engineering
& Mathematical Sciences
University of the Westof England
Bristol BS16 1QY

Toby O'Hara
Faculty of Computing, Engineering
& Mathematical Sciences
University of the West of England
Bristol BS16 1QY

Abstract

Learning Classifier Systems traditionally use a binary representation with wildcards added to allow for generalizations over the problem encoding. However, the simple scheme can be limiting in complex domains. In this paper we present results from the use of neural network-based representation schemes within the accuracy-based XCS. Here each rule's condition and action are represented by a small neural network, evolved through the actions of the genetic algorithm. After describing the changes required to the standard production system functionality, optimal performance is presented using multi-layered perceptrons to represent the individual rules. Results from the use of fuzzy logic through radial basis fuction networks are then presented. In particular, the new representation scheme is shown to produce systems where outputs are a function of the inputs.

1 INTRODUCTION

Since their inception Learning Classifier Systems (LCS) (Holland 1986) have been compared to neural networks, both conceptually (e.g. Farmer 1989) and functionally (e.g. Davis 1989, Dorigo & Bersini 1994, Smith & Cribbs 1994). In this paper we present a way to incorporate the neural paradigm into the accuracy-based XCS (Wilson 1995). LCS traditionally incorporate a binary rule representation, augmented with 'wildcard' symbols to allow for generalizations. This can become limiting in more complex domains (e.g. see (Schuurmans & Schaeffer 1989) for early discussions). Recently, a number of investigations have made use of other rule representations, including real numbers (Wilson 2000), messy GAs (Lanzi 1999a), logical S-expressions (Lanzi 1999b), and those where the output is a function of the input, including numerical S-expressions (Ahluwalia & Bull 1999) and fuzzy logic (e.g. Valenzuela-Rendon 1991).

We present a neural network-based scheme where each rule's condition and action are represented by a neural network. The weights of each neural rule being concatenated together and evolved under the actions of the genetic algorithm (GA)(Holland 1975). The approach is closely related to the use of evolutionary computing techniques in general to produce neural networks (see (Yao 1999) for an overview). In contrast to most of that work, an LCS-based approach is coevolutionary, the aim being to develop a number of (small) cooperative neural networks to solve the given task, as opposed to the evolution of one (large) network. That is, a decompositional approach to the evolution of neural networks is proposed. Moriarty and Miikulainen's SANE (1997) is most similar to the work described here, however SANE coevolves individual neurons to form a large network rather than small networks of neurons as rules.

2 X-NCS: A NEURAL LCS

2.1 XCS

In XCS rule-fitness for the GA is not based on rule predictions but on the accuracy of the predictions. The intention being to form efficient generalizations and a complete and accurate mapping of the search space (rather than simply focusing on the higher payoff niches in the environment).

On each time step match sets [M] are created. A system prediction is then formed for each action proposed by the rules in [M] according to a fitness-weighted average of the predictions of the rules. The system action is then selected, typically either deterministically (exploit) or randomly (explore). An action set [A] is then formed, the appropriate system output given and a reward may or not be received. If [M] is empty covering is used.

Reinforcement in XCS consists of updating three parameters, Error (E), Prediction (p), and fitness (F) for each appropriate rule. Each is updated every time it belongs to $[A_{-1}]$ or [A] if it is a single step problem.

XCS uses a niche-GA (Booker 1985); the GA acts in action sets [A]. Two rules are selected based on fitness. In this paper we use a fixed size rule-base N; varying N as in (Wilson 1995) is not incorporated here. Rule replacement

is based on the estimated size of each match set a rule participates in with the aim of balancing resources across niches. The GA is triggered (see also (Booker 1989)) within a given match set if the number of time steps since its last invocation in that set passes a fixed threshold, based on the average time-stamp of the rules. Typically this parameter is set to 25.

The reader is referred to (Butz & Wilson 2001) for full details of XCS. Wherever possible parameter values are in line with those used in (Wilson 1995) to facilitate comparisons with XCS using a ternary alphabet. In practice all parameter values lay within those used in (Wilson 1995) apart from two areas: population size and mutation rate which, as they relate to the higher number of real number genes in the genotype, are higher; and in function approximation, the relative value of the error parameter is a percentage value rather than being fixed. Further, we don't incorporate subsumption or maintain a rule for each action in a given [M].

All results in this paper are the average of ten runs.

2.2 NEURAL RULE REPRESENTATION

Each traditional condition-action rule is replaced by a single, fully connected neural network. All rules have the same number of nodes in their hidden layers (simplest case (Bull 2001)) and one more output node than there are possible actions. All weights are randomly initialized in the range { –1.0, 1.0 }, concatenated together in an arbitrary order and thereafter determined solely by the GA here.

The production system cycles through the same input-match-action-update cycle as the LCS, XCS in this case. However, since all rules explicitly 'see' all inputs, unlike the traditional scheme whereby defined loci can exclude certain rules from certain match-sets, the extra output node is added. This is used to signify membership of a given match-set. After the presentation of an input, each neural network rule produces a value on each of its output nodes in the appropriate manner, e.g. feedforward. If the extra 'not match-set member' node has the highest output value, the rule does not form part of the resulting match-set. In all other cases the rule forms part of the match-set, proposing the action corresponding to the output node with the highest activation. This matching procedure is repeated for all rules on each cycle.

Rule discovery operates in the same way as usual for XCS with real numbers (Wilson 2000). Hence the mutation operator is altered to adjust gene values using a normal distribution; small changes in weights are more likely than large changes upon satisfaction of the mutation probability (μ). The cover operator is altered such that when the match-set is empty, random neural networks are created until one gives its highest activation on an action node for the given input.

Results from using multi-layered perceptrons (MLPs) are now presented in well-known single-step and multi-step tasks. All nodes used a sigmoid transfer function.

3 A SINGLE-STEP TASK: 6-BIT MULTIPLEXER

We have tested the new rule representation on the two tasks used in (Wilson 1995), the first of which is a 6-bit version of the well-known, single-step multiplexer task. These boolean functions are defined for binary strings of length $l = k + 2^k$ under which the first k bits index into the 2^k remaining bits, returning the indexed bit.

In order to make analysis easier for each neural net an equivalent classifier was produced and recorded, though it must be emphasized that it played no part in the XCS processing, and was produced to measure the generality or specificity of the particular neural rule.

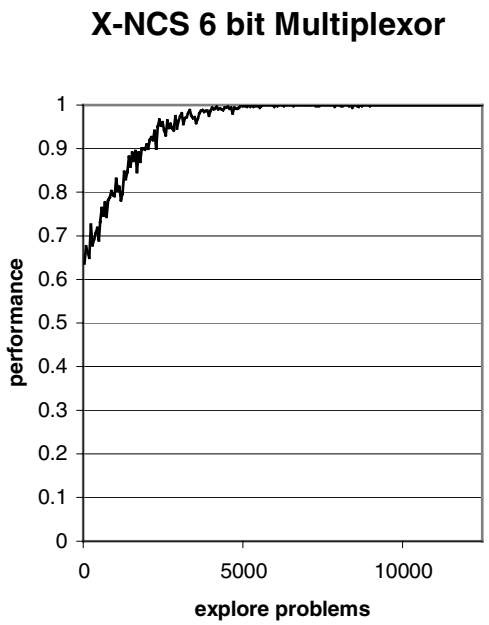


Figure 1: Performance of accuracy-based neural classifier system on the 6-bit multiplexer (l=6).

Figure 1 shows the results of using X-NCS on the single-step problem, averaged over ten runs, with all parameters as presented in (Wilson 1995) apart from the population size and mutation. That is, N=800, μ=0.08, β=0.2, ϕ=0.5, α=0.1, χ=0.8, θ=10, δ=0.1, $p1$=10.0, $F1$=10.0, ε1=0.0. As in (ibid.), payoff is given in 100 increments from 300/0 for each classification. Rules contain five nodes in their hidden layer.

From Figure 1 it can be seen that using the neural representation requires around 5000 exploit problems to solve the task, roughly equivalent to the binary

representation (Wilson 1995). However, that an overhead may be incurred for a more complex representation may perhaps expected for such simple tasks. Analysis of the resulting rule-bases shows that, as well as the usual rules which match multiple inputs and propose a single action at a given payoff prediction level, multiple action rules emerge. That is*, for a given prediction level, accurate rules are evolved which suggest different actions depending on the input*.

4 A MULTI-STEP TASK: WOODS 2

Wilson (1995) presented the multi-step, and hence delayed reward, maze task Woods 2 to test XCS. Woods 2 is a toroidal grid environment containing two types of food (encoded 110 and 111), two types of rock (encoded 010 and 011) in regularly spaced 3 by 3 cells, and free space (000) (see (ibid.) for full details).

The learner is positioned randomly in one of the blank cells and can move into any one of the surrounding eight cells on each discrete time step, unless occupied by a rock. If it moves into a food cell the system receives a reward from the environment (1000) and the task is reset, i.e. food is replaced and the learner randomly relocated. On each time-step the learning system receives a sensory message, which describes the eight surrounding cells, ordered with the cell directly north and proceeding clockwise around it.

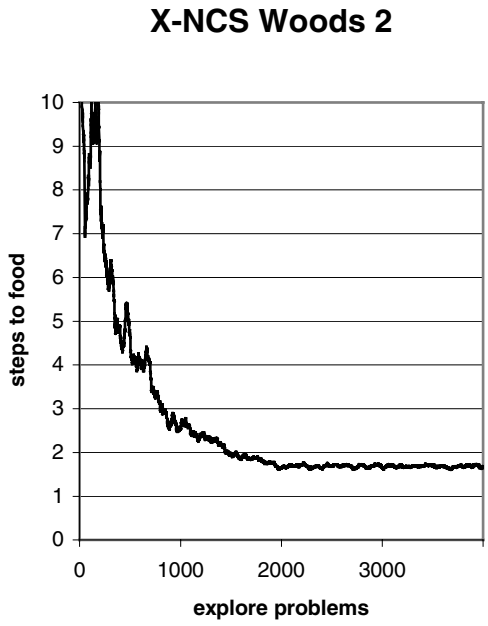


Figure 2: Performance of the accuracy-based neural classifier system in Woods 2.

Here, as in (Wilson 1995), the trial is repeated 8000 times, half explore and half exploit, and a record is kept of the moving average (over the previous 50 exploit trials) of how many steps it takes the system to move into a food cell on each trial.

Figure 2 shows results from using X-NCS in Woods 2 with all parameters as presented in (Wilson 1995) except, more rules and more mutation are used. That is, N=1600, μ=0.08, β=0.2, γ=0.71, ϕ=0.5, α=0.1, χ=0.8, θ=25, δ=0.1, p_1=10.0, F_1=10.0, ε_1=0.0. Rules again contained five hidden nodes. It can be seen that it takes the system around 2,000 explore problems, again roughly equivalent to the traditional encoding, to reach optimal performance (1.7 steps to food).

Analysis of the resulting rule-bases shows that neural rules emerge which have no error and produce different actions depending upon the input. Unlike in the multiplexer problem, we find that for some payoff levels these multi-action rules are more numerous than the equivalent single action rules. We presume that, if left to run for longer, the system would converge on a single neural rule for each payoff level; *maximal generalizations would be produced in both the condition and action space.*

As noted above, XCS usually forms generalizations for each action at each level of payoff. Within traditional reinforcement learning (Sutton & Barto 1998) a neural network is often used to produce generalizations for each possible action, where the networks are trained using gradient descent techniques. Under the scheme proposed here, X-NCS forms generalizations at a level between these two extremes using the GA to produce the neural networks. An advantage of this scheme over the other two is its ability to work with continuous action spaces. An application which exploits this last aspect of the representation scheme in a single-stepped task is now presented.

5 FUNCTION APPROXIMATION

It is well-known that multi-layered perceptrons with an appropriate single hidden layer and a non-linear activation function are universal classifiers (e.g. Hornick et al. 1989). Until recently LCS had not been used to solve tasks of the form $y = f(x)$ since their traditional representation scheme does not lend itself to such classes of problem. Fuzzy Logic LCS (see (Bonarini 2000) for an overview) represent, in principle, a production system-like scheme which can be used for such tasks but this remains unexplored. Ahluwalia and Bull (1999) presented a simple form of LCS which used numerical S-expressions for feature extraction in classification tasks. Here each rule's condition was a binary string indicating whether or not a rule matched for a given feature and the actions were S-expressions which performed a function on the input feature value. Most recently, Wilson (2001) has presented a form of XCS, termed XCSF, which uses piecewise-linear approximation for such tasks; using only explore trials all matching rules update their parameters, where such trials are run consecutively as a training period.

We have tested the neural rule representation for tasks of the form $y = f(x)$, where both x and y are real numbers between 0.0 and 1.0. The implementation estimated two functions, x-squared and a six variable root-mean-square. However, unlike the above mentioned work, the system requires very few changes to the design of the standard XCS system.

5.1 MODIFICATIONS TO X-NCS FOR FUNCTION APPROXIMATION

5.1.1 Processing of Real Numbers

The real number inputs were scaled between 0.4 and 0.8 to accommodate the lack of discrimination of the upper and lower end of the sigmoid function, as is usual in the use of MLPs. Output layer nodes are now linear.

5.1.2 Changes to Error Threshold Processing and System Error

In standard XCS, the error threshold ε_0 is a fixed fraction of the payment range. However with function approximation across a continuous (action) range, a fixed value may result in very inaccurate classifiers at the bottom end of the input range. It was therefore decided that ε_0 should be variable to enable the accuracy, and hence fitness, of the classifiers across the range to be equivalent. The variable value was chosen as the percentage of the target value at any particular point. The percentage chosen was 1% so, for example under x-squared, if the input was 0.3 the target output value ($f(x)$) would be 0.09. Here the required accuracy ε_0 would be 0.0009 and so classifiers that predicted within the range 0.0891 to 0.0909 would be given an accuracy of 1.0. In the same way, when the performance of the system is measured, the system error was calculated by taking the absolute difference between the target value and the prediction of the selected classifier, and dividing this by the target value, i.e. the system error is the percentage error between the target and the prediction value.

5.1.3 Match Set and Action Set Processing

The rule prediction value is taken from one output node of the individual's neural network. The selection of the match set is similar to before (Section 2), the only difference being that the 'not match-set member' output node merely has to have a positive value, rather than a value less than the primary output node, as above. The aim being to reduce the complexity of the task faced by individual rules.

In exploration all members of the match set are updated and rule discovery invoked if appropriate as per standard XCS. In XCS under exploitation, all classifiers which advocate the same action are put into the same set [A]. The chosen action set is the one which has the highest fitness weighted prediction. For function approximation we are looking for the rule whose prediction is most accurate, i.e. has the least error, and hence taking the classifier with the highest fitness weighted prediction would be inappropriate. Instead, the counterpart of prediction for such tasks is chosen, i.e. rule error, and so we choose the rule with the lowest value of error divided by fitness.

It was also found that for these function approximation tasks a biased uniform crossover operator (75%) appeared to give slightly better results than the single point crossover operator used above. This aspect of the system remains open to future investigation.

5.1.4 Rule Updating

The prediction value for each rule is taken as the value of the output of the neural network, i.e. the prediction value of the classifier can change at each iteration. By contrast, the error value of a rule is determined as per standard XCS. Accuracy is determined in the standard XCS way except, as mentioned above, the accuracy criterion is taken as a percentage of the current target value. Fitness is again calculated in the standard XCS way.

Thus the output value for a particular rule will change for each different input value. For example, for problem n with input value 0.3 -> prediction 0.0891, but problem n+1 with input 0.4 -> prediction 0.160. However the error value for each accurate classifier, although it varies as the predictions can deviate from their respective targets, is a small value that oscillates according to ε_0.

5.2 RESULTS FOR $Y=X^2$

In this task training consists of (alternating) 50,000 explore trials and 50,000 exploit trials each presenting a random input in the range [0.0, 1.0] scaled as mentioned above.

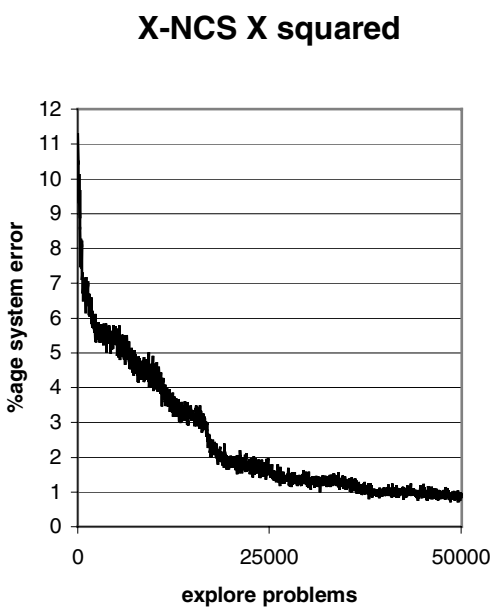


Figure 3: X-NCS on the x-squared function.

Figure 3 shows the performance of the accuracy-based neural classifier system on the x-squared function, averaged over ten runs, with a runnning average over the previous fifty exploit trials. The parameters used were: N=1000, β=0.2, ϕ=0.5, μ=0.03, α=0.1, χ=0.8, θ=10, δ=0.1, p_1=10.0, F_1=10.0, ε_1=0.01. Rules contained five hidden layer nodes. For the last 500 problems X-NCS is run in test mode and hence under the exploit scheme; after training with randomly generated examples the performance of the resulting system was tested using a number of unseen randomly generated examples.

From Figure 3 it can be seen that using the neural representation requires around 40,000 explore problems to solve the task, i.e. for the accuracy of the approximations to fall within 1% of the real $f(x)$.

Analysis of the resulting systems shows that better performance is achieved when one neural network emerges to cover the whole problem space, rather than through the co-operative sets seen above. The reasons for this appear two-fold: MLPs attempt to form global models by approximating between known data points; and the niche-based scheme of XCS encourages maximally general rules through increased chances to reproduce. This aspect of X-NCS will be returned to.

5.3 RESULTS FOR ROOT-MEAN-SQUARE

We have also examined the performance of the system on functions which contain more than one variable. Wilson (2001) presented a general, multi-dimensional function of the form $y = [(x1^2 + \ldots + xn^2) / n]$ ½. We have used this "root mean squared" function with n=6, where training was identical to that of the x-squared task above.

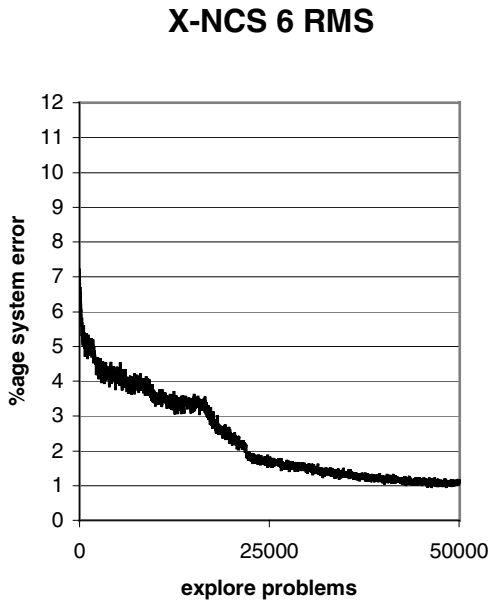


Figure 4: X-NCS on the 6 variable rms function.

The parameters used were: N=1000, β=0.2, ϕ=0.5, μ=0.09, α=0.1, χ=0.8, θ=10, δ=0.1, p_1=10.0, F_1=10.0, ε_1=0.01.

From Figure 4 it can be seen that using the neural representation requires around 50,000 explore problems to solve the task to an accuracy of 1%. As with the x-squared function, the most accurate solutions came from those in which one classifier covered the whole input range.

6 X-NFCS: A NEURO-FUZZY LCS

Radial basis function neural networks (RBFs) (e.g. Poggio & Girosi 1990), in contrast to MLPs, construct local function approximations using Gaussian functions processed in the hidden layer of the network. It was noted above (Sections 3 and 4) that in the discrete action tasks the MLP-based neural system formed traditional LCS coevolutionary solutions, whereas with continuous actions a single rule/network emerged. Hence the use of RBFs in X-NCS seems more likely to exploit the system's coevolutionary nature. There is another potential benefit to the use of RBFs.

The similarity between RBF networks and fuzzy rule-based systems is discussed in (Jang & Sun 1995). Fuzzy rule sets consist of membership functions over appropriate universes of discourse for input and output variables and rules which define input-output relations. The Gaussian functions of an RBF can be seen as fuzzy membership functions and the hidden layer nodes the fuzzy rules. In general, the benefits from combining neural computing with fuzzy logic are potentially large (see (Tsoukalas & Uhrig 1997) for an introduction). In this context it also avoids the possible need to alter the reinforcement process (see (Bonarini 2000) for discussions).

GAs have been used to evolve RBFs as they have MLPs. The most similar approach to that proposed here is Whitehead and Choate's (1995) scheme whereby the individual members of the population are the basis functions of a single network and heuristics tackle the competitor/cooperator problem.

Genomes are again strings of real numbers: the positions of the basis function centres, widths and weights of fully connected networks are concatenated in an arbitrary order to form the encoding. Output nodes (two) are again linear. Carse et al. (e.g. 2001) have proposed a crossover operator for fuzzy sets which alleviates the permutations (Radcliffe 1990) problem that can arise under the evolution of neural networks. That is, different genotypes can give the same phenotype and hence crossover may disrupt useful structures. The fuzzy logic crossover operator works in the input space rather than by postion on the genome. As with the MLPs, and perhaps due to the niche GA of XCS, the permutations aspect of the concatenated weights encoding does not appear to have been significant in the tasks explored here. All other system functionality is the same as in Section 5 – a

positive response on the extra node means a rule doesn't join a given [M] and percentage error is used.

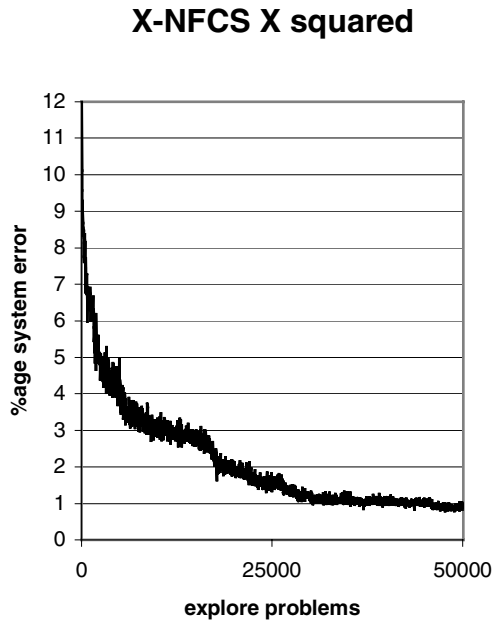


Figure 5: X-NFCS on the x-squared function.

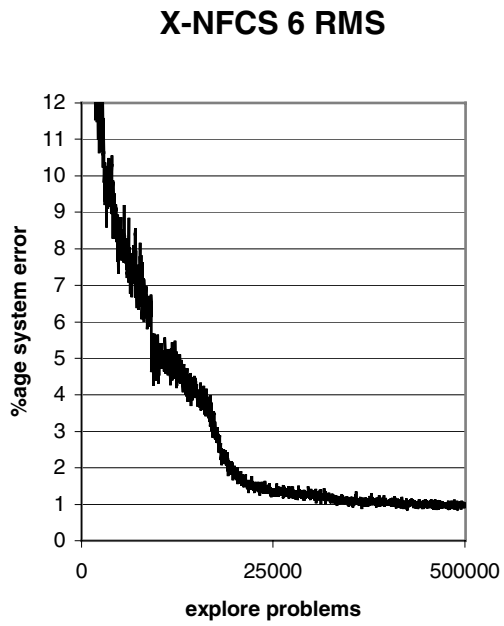


Figure 6: X-NFCS on the 6 variable rms function.

Figure 5 shows the performance of the accuracy-based neuro-fuzzy classifier system on the x-squared function, averaged over ten runs, with a runnning average over the previous fifty exploit trials. All parameters are as in section 5.2. It can be seen that the function is learnt to 1% accuracy around 30,000 explore problems. That is, a greatly reduced training period is required using the neuro-fuzzy system (compare with Figure 3). Analysis of the resulting systems shows that coevolutionary solutions appear, i.e. different rules emerge to handle different regions of the input space, together covering the total problem space.

Figure 6 shows the performance of X-NFCS on the six variable rms function. All parameters are as in section 5.3, but it was found necessary to use ellipsoid radial basis functions. Here, each input node maintains a radius for its Gaussian for each input node. It can be seen that accurate performance is obtained after 40,000 explore trials. Analysis again shows that a number of rules are used in the evolved systems and that the decompositional approach led to a reduced training period (compare with Figure 4).

Results (not shown) from the discrete action tasks of sections 3 and 4 also show optimal performance using RBFs, with learning times similar to the MLP-based system.

7 CONCLUSIONS

In this paper we have presented results from using a neural rule representation scheme within an accuracy-based learning classifier system. The effective combination of evolutionary computing and neural computing has long been an aim of machine learning (e.g. Belew et al 1989). It is our aim to exploit the coevolutionary and accuracy processes of XCS to realize such systems. Hopefully, this will also ease the use of LCS in more complex problem domains.

We are currently examining the use of the system for more complex tasks, with discrete or continuous action spaces, both single-step and multi-step. For the latter we are also exploring the use of recurrent connections with the neuro and neuro-fuzzy systems for non-Markov domains (after (Bull & O'Hara 2001)).

References

Ahluwalia, M. & Bull, L. (1999) A Genetic Programming Classifier System. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds*) Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99.* Morgan Kaufmann, pp11-18.

Belew, R.K., McInerney, J. & Schraudolph, N.N. (1989) Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. In C.G. Langton, C. Taylor, J.D. Farmer & S. Rasmussen (eds) *Artificial Life II,* Addison-Wesley, pp511-548.

Bonarini, A. (2000) An Introduction to Learning Fuzzy Classifier Systems. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp83-106.

Booker, L. (1985) Improving the Performance of Genetic Algorithms in Classifier Systems. In J.J. Grefenstette (ed.) *Proceedings of the First International Conference on Genetic Algorithms and their Applications,* Lawrence Erlbaum Assoc., pp80-92.

Booker, L. (1989) Triggered Rule Discovery in Classifier Systems. In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, pp265-275.

Bull, L. (2001) A Brief note on the use of Constructivism in NCS. *UWE Learning Classifier Systems Group Technical Report UWELCSG01-006*. Available from http://www.cems.uwe.ac.uk/lcsg.

Bull, L. & O'Hara, T. (2001) NCS: A Simple Neural Classifier System. *UWE Learning Classifier Systems Group Technical Report UWELCSG01-003*. Available from http://www.cems.uwe.ac.uk/lcsg.

Butz, M. & Wilson, S.W. (2001) An Algorithmic Description of XCS. *Advances in Learning Classifier Systems: Proceedings of the Third International Conference – IWLCS2000*. Springer, pp253-272.

Carse, B, Fogarty, T.C, Patel, & Sullivan, J (2001) Evolution and Learning in Radial Basis Function Neural Networks- A Hybrid Approach. In M.J V. Honavar & K. Balakrishnan (eds) *Advances in the Evolutionary Synthesis of Intelligent Agents.* MIT Press, pp247-272.

Davis, L. (1989) Mapptin Neural Networks into Classifier Systems. In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, pp375-378.

Farmer, D. (1989) A Rosetta Stone for Connectionism. *Physica D* 42:153-187.

Holland, J.H.(1975) *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Holland, J.H. (1986) Escaping Brittleness. In R.S. Michalski, J.G. Carnoell & T.M. Mitchell (eds) *Machine Learning: An Artificial Intelligence Approach 2*. Morgan Kaufmann, pp48-78.

Hornick, K., Stinchombe, M. & White, H. (1989) Multiayer Feedforward Networks are Universal Approximators*. Neural Networks 2*(5): 359-366.

Jang, J-S & Sun, C-T. (1995) Neuro-fuzzy Modeling and Control. *Proceedings of the IEEE* 83(3): 378-406.

Lanzi, P-L (1999a) Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds*) Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99*. Morgan Kaufmann, pp11-18.

Lanzi, P-L (1999b) Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds) *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-99*. Morgan Kaufmann, pp11-18.

Moriarty, D.E. & Miikulainen, R. (1997) Forming Neural Networks Through Efficient and Adaptive Coevolution. *Evolutionary Computation* 5(2): 373-399.

Poggio, T. & Girosi, F. (1990) Networks for approximation and learning *Proceedings of the IEEE*, 78:1481-1497.

Radcliffe, N.J. (1990) Genetic Set Recombination and its Application to Neural Network Topology Optimisation. Technical Report EPCC-TR-91-21, University of Edinburgh, Scotland.

Schuurmans, D. & Schaeffer, J. (1989) Representational Difficulties with Classifier Systems. In J.D. Schaffer (ed.) *Proceedings of the Third International Conference on Genetic Algorithms,* Morgan Kaufmann, pp328-333.

Smith, R.E. & Cribbs, H.B. (1994) Is a Learning Classifier System a Type of Neural Network*? Evolutionary Computation* 2(1): 19-36.

Sutton, R. & Barto A. (1998) *Reinforcement Learning*. MIT Press.

Tsoukalas, L.H & Uhrig, R.E. (1997) *Fuzzy and Neural Approaches in Engineering*. Wiley.

Valenzuela-Rendon, M. (1991) The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In L. Booker & R. Belew (eds*) Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp346-353.

Whitehead, B.A. & Choate, T.D. (1994) Evolving Space-filleing Curves to Distribute Radial Basis Functions over an Input Space*. IEEE Transactions on Neural Networks* 5(1): 15-23.

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

Wilson, S.W. (2000) Get Real! XCS with Continuous-Valued Inputs. In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Learning Classifier Systems: From Foundations to Applications*. Springer, pp209-222.

Wilson, S.W. (2001) Function Approximation with a Classifier System. In L. Spector. E.D Goodman, A. Wu, W.B. Langdon, H-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon & E. Burke (eds*) Proceedings of the Genetic and Evolutionary Computation Conference – GECCO-2001*. Morgan Kaufmann, pp974-984.

Yao, X. (1999) Evolving Artificial Neural Networks. *Proceedings of the IEEE* 87(9): 1423-1447.

XCS Applied to Mapping FPGA Architectures

Martin Danek

Dept. of Computer Science and Engineering
Czech Technical University in Prague
Faculty of Electrical Engineering
Czech Republic
Email: danek@sun.felk.cvut.cz

R. E. Smith

Intelligent Computer Systems Centre
University of The West of England, Bristol
Computing, Engineering and
Mathematical Sciences Faculty
United Kingdom
Email: robert.smith@uwe.ac.uk

Abstract

This paper considers the application of XCS to the complex, real-world problem of mapping Boolean networks to technology-specific layout of field programmable gate arrays (FPGAs). The mapping is formulated as a temporal task, where the XCS's actions are to create blocks (based on an abstract Boolean network) that can be placed in the FPGA, one-at-a-time. Despite the complexity of this task, we demonstrate that the system is effective. We demonstrate the transfer of knowledge stored in an XCS rule set from a small FPGA mapping problem, to a larger problem. We present a novel technique that utilizes a problem-specific finite state machine and two populations of classifiers to treat the problem hierarchically. Final sections of the paper discuss implications and future directions for this work.

1 INTRODUCTION

Field-programmable gate arrays (FPGAs) are semi-custom VLSI circuits that were first introduced by the Xilinx company in 1984. They usually consist of a two-dimensional matrix of configurable logic blocks (CLBs) surrounded by special input-output blocks (IOBs) on the perimeter of the CLB matrix. An important part of the chip is formed by a programmable interconnect that can be used to connect inputs and outputs of logic blocks to form a desired circuit. The granularity of the logic blocks differs from fine-grain (universal two-input logic gates in the Xilinx XC6200 family) to coarse-grain (two four-input and one three-input cascaded look-up tables (LUTs) in the Xilinx XC4000 family).

FPGAs are favored for their short design cycle (see Figure 1), since one design cycle (one design implementation) takes hours rather than days (in the case of mask-programmable gate arrays), or months (in the case of application-specific integrated circuits, ASICs). This together with their low cost makes them suitable both for low-volume production and for prototyping of ASICs.

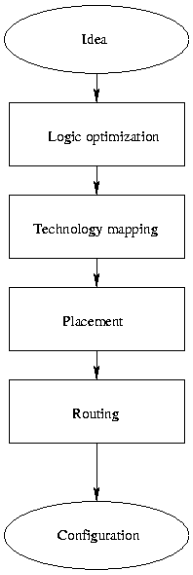


Figure 1: A typical FPGA design cycle.

Current field programmable gate arrays are very complex devices. Design software is needed to assist the user in implementing a circuit. This software usually consists of several subsequent optimization routines that transform the information about the circuit to be implemented (the *netlist*) to a more device-specific form.

These routines are usually referred to as

- the *mapper*,
- the *placer* and
- the *router*.

The software-assisted design procedure is typically as follows:

1. A design is specified by an abstract Boolean network (ABN) of logic gates and flip-flops.
2. The mapper performs a device-independent logic optimization. This transforms logic functions so as to obtain their minimal form.

3. Then the mapper maps the design to the target technology, which means that the gates in the netlist that represent the minimized logic functions and flip-flops are translated to blocks that are supported by the target FPGA (up to five-input LUTs and D-type flip-flops (DFFs) in the case of XC4000 devices).

4. The mapper forms groups of these blocks that fit into one CLB. Each CLB in the XC4000 devices can implement two four-input LUTs (usually referred to as F-LUT and G-LUT), one three-input LUT, (usually referred to as H-LUT), and two DFFs, (usually referred to as DXFF and DYFF). See Figure 2.

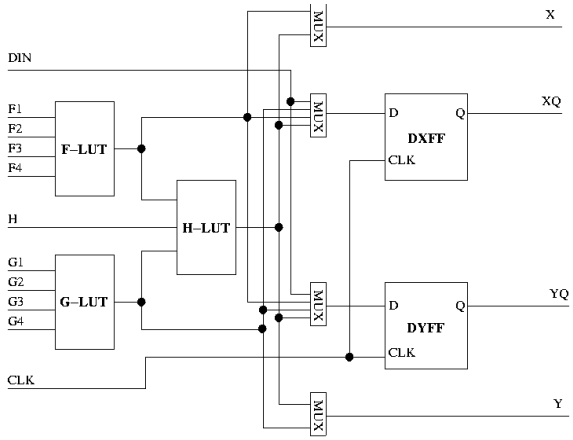


Figure 2: An internal structure of a configurable logic block (XC4000).

5. The placer assigns these groups (in fact, CLBs) physical positions in the CLB matrix of the target FPGA.

6. The router determines the way the CLBs are connected by universal wire segments that implement signal networks from the netlist. This consists of assigning wire segments to individual signal networks and of connecting these segments together.

7. As a final step, based on all this information, a device configuration bit stream is generated that can be downloaded to the FPGA, such that it is programmed to the desired function.

2 THE MAPPING TASK

The mapping phase takes an interconnection of abstract operators (the subject Boolean network) and generates an interconnection of logic cells selected from a given library. For instance, in the case of the Xilinx XC4000 devices, these are up to four-input look-up tables and edge-triggered D-type flip-flops.

The task of technology mapping in the case of LUT-based FPGAs is to find a set of clusters of technology dependent units that cover the whole Boolean network. A good cluster selection mechanism should take into account both the internal structure of the configurable logic block (CLB) of the target device (for an example, see [15]) and routing delays, in order to achieve good area and performance results.

The mapping task is an NP-complete problem [8]. The prevailing way to tackle it is to implement a heuristic algorithm, based on a previously gained human knowledge of the target architecture. This approach may be limiting for more complex devices, because they may contain features not recognizable at first sight. In addition, there is the question of considering signal delays in the mapping phase. Up-to-date mappers approximate delays with a unit delay per one level of logic, or they use signal delay values that were generated in previous iterations [4]. The unit-delay approach is very fast and is optimal in situations where the design topology is a sort of a planar graph. The iterative approach is time-consuming, because it requires several mapping runs interleaved with running placement and routing algorithms, and it does not guarantee the exactness of delay estimations, because they are reliable only in cases when small modifications take place.

The set of targeted logic cells in the mapping phase can be given either by an enumeration (i.e., as a set of standard cells (functions) for ASICs), or by a prescription (i.e., by specifying the maximal number of inputs a function may have, given that the function itself can be anything).

The first case is called *library-based mapping*, and it is suitable for all technologies that have fixed and relatively small libraries. For FPGAs, and especially for FPGAs based on look-up tables, this approach is not feasible, as the number of functions (i.e., library elements) grows rapidly with the increasing number of LUT inputs.

Currently, *prescription-based mapping* is very popular for LUT-based FPGAs. The drawback is that while known methods permit only simple prescriptions (commonly a single LUT); the logic blocks of existing FPGAs can be configured for more complex structures that are comprised of multiple LUTs. FPGAs with this ability (for example the Xilinx XC4000) are called *heterogeneous*.

The motivation for using heterogeneous FPGAs is a compromise between predictability of future routing (by increasing the number of routes with predictable delays - the ones hidden inside configurable logic blocks) and wasting logic that cannot be used (due to a fixed internal structure of logic blocks).

A mapping algorithm for heterogeneous FPGAs should combine the properties of both the approaches outlined above. Fortunately, the number of distinct configurable structures (regardless of the actual LUT contents) is conveniently small. The description can then be characterized as an enumeration of prescriptions. Although the algorithm that generates all clusters to cover the abstract Boolean network (see Figure 3) is relatively

straightforward, the number of generated clusters increases with the complexity of physical cells and the proper cluster selection constraints become more complicated [10].

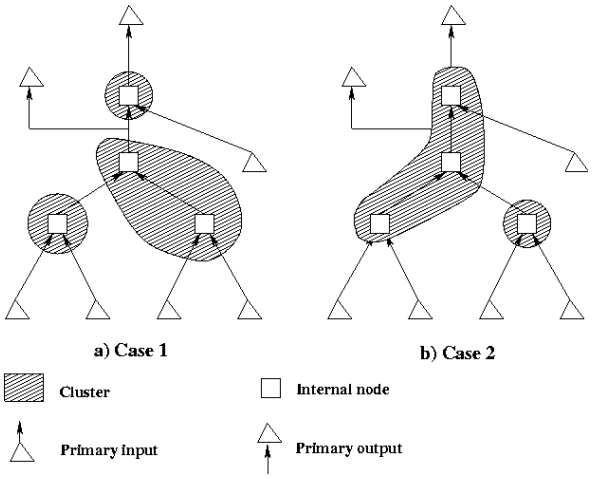


Figure 3: Covering an ABN with clusters - two cases.

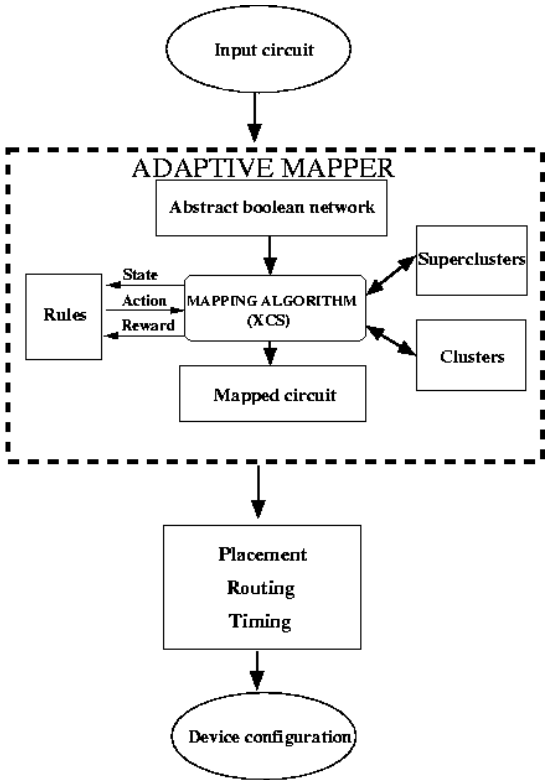


Figure 4: The XCS-based adaptive mapper.

3 ADAPTIVE MAPPING WITH XCS

This paper presents a rule-based, single-pass, adaptive mapper (see Figure 4) based on Wilson's XCS classifier system [12]. The XCS system was chosen due to its relative simplicity and ease of analysis.

It is intended that the mapper will operate by first being trained on benchmark circuits, such that it evolves a set of general mapping rules that perform well for any design, then using these pre-evolved rules to map a different, user-specified design in a single pass.

In the implementation presented here, the classifier system calls elementary mapping actions, and it attempts to evolve sequences of rules that lead to minimal CLB usage and small critical signal path delays. The approach used is similar to the FSM worlds used by Barry [2], but here the situation is more complicated. The underlying graph is not linear, but it is a directed tree with much longer start-to-terminal node distances, and some mapping actions group several nodes together and thus directly modify the environment. If all goes well, the adaptive nature of the classifier system should ensure that the final rules take into account global properties of the FPGA chip. Initially, the mapper is aware only of local properties of the chip - the internal architecture of CLBs. At the end of the run, we intend to be able to assemble efficient sequences of actions for the mapping task.

4 IMPLEMENTATION DETAILS

The structure of the XCS adaptive mapper is outlined in the following sections.

4.1 CONDITIONS

At any given stage of the mapping process, the state of the problem is described to the XCS by a *sense vector,* against which classifier conditions are mapped.

The sense vector used here consists of equal groups of binary flags, with each group representing the current CLB to be processed (the CLB that the XCS will place next), and CLBs connected to the inputs and outputs of that CLB. At present, only the maximum of four input and four output blocks are considered. Each group contains the flags representing the following binary characteristics of the associated block:

- addable to the current F-LUT, G-LUT, H-LUT,
- addable to the current DXFF, DYFF,
- connected to the current F-LUT, G-LUT, H-LUT (‘current F-LUT’ means F-LUT of the CLB that will be generated by the next GenCLB action, etc.)
- connected to the current DXFF, DYFF.

The sense vector also contains an extra group of flags that validate the previous groups, and that reflect the utilization of LUTs and DFFs of the CLB currently under construction (see Figure 5).

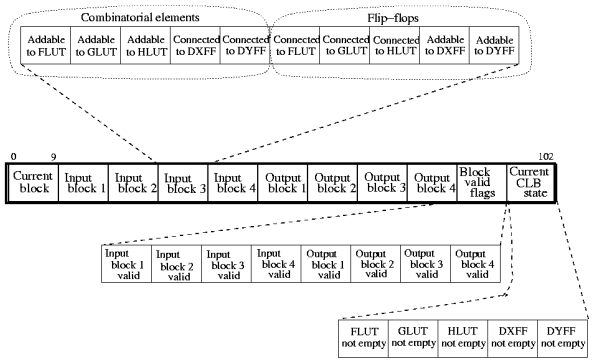


Figure 5: Diagram of sense vector that represents the problems state to the classifier system.

The length of the resulting sense vector is 103 bits. Although this represents a large state space, the advantage of such a coding is the clear meaning of genetic operations performed by the classifier system.

4.2 ACTIONS

The XCS adaptive mapper has three groups of mapping actions:

- actions that construct clusters (that stand for look-up tables (LUTs) or flip-flops) and assign them to the current supercluster (a supercluster is equivalent to a CLB),
- actions that generate the current supercluster, and
- actions that modify the current position in the netlist and an auxiliary *do nothing* action.

The first group contains actions:

- expand F-LUT, G-LUT, H-LUT, which assign a block to an empty cluster (LUT) or add a block to an existing cluster,
- assign a flip-flop to DXFF, or DYFF.

The only action in the second group takes clusters that were assigned to the current supercluster, generates the supercluster and empties all clusters:

- generate a CLB.

The third group consists of eight actions that implement different *move backward* and *move forward* commands, and one *do nothing* action.

The total number of actions available to the mapper is 15. The XCS software uses only the mutation operator for actions; and it randomly generates a new action number from the 15 alternatives.

4.3 REWARDS

The processing of rewards is based on reinforcement learning techniques [1]. To formulate the reward function correctly, it is necessary to declare the goals of the task correctly. The final goal is to generate efficiently a high-performance mapping of a design, which means:

- to use CLBs to their capacity
- to use as few CLBs as possible,
- to minimize the critical path delay, and
- to accomplish it in as few steps as possible.

This suggests that the reward should reflect the decrease in the number of CLBs used in the design and the decrease in the critical path delay. Each (unsuccessful) execution of an action should be given a negative reward.

A closer analysis of the interaction of the actions suggests the introduction of another goal: using CLBs to their capacity. Therefore, the reward received on a successful completion of an action was formulated as a weighted combination of factors:

$$reward = \left(w_{num} * \Delta numCLBs\right) + \left(w_{delay} * \sum\left(\Delta path\ delays\right)\right) + \left(w_{usage} * CLBusage\right)$$

where $\Delta numCLBS$ is the change in the number of CLBs used by the design at a time step t, each $\Delta pathdelay$ is the change in the delay of a single (two-point) signal path delays at a time step t (and the sum is taken over all such signal paths), and *CLBusage* is the utilization of the possibly generated CLB (or zero) (see Figure 6). To evaluate the reward function the number of CLBs (each unassigned block is declared to occupy one CLB) and actual values of path delays are calculated after an execution of each action. So far only a simple unit-delay model (see Section 2) is used, it is planned to replace it with a more realistic estimation of individual signal network delays in the future.

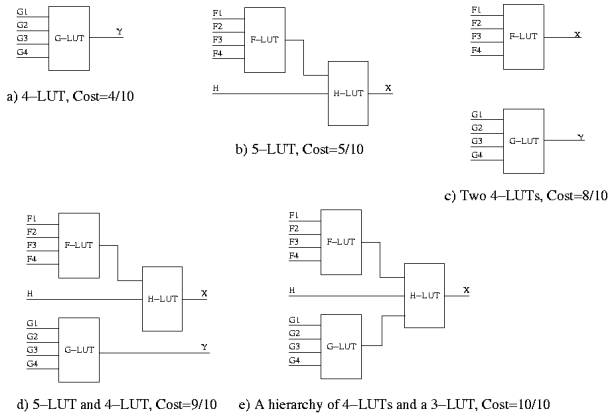


Figure 6: Different CLB configurations and their cost.

5 EXPERIMENTS

The target architecture for all runs presented here is the XC4000 CLB, the delay estimation used the unit delay model, and the evaluation was done for the LGSynth91 (also known as MCNC91) benchmark circuits [16]. The benchmarks come in the widely accepted BLIF format

and are translated to the Xilinx XNF format to be compatible with the Xilinx tools.

Parameter values used in experiments are shown in Table 1:

Parameter	Value
w_{num}	7
w_{delay}	12
w_{usage}	4
β (learning rate)	0.2
γ (discount factor)	0.8
Exploration policy	Constant
Exploration probability	0.2

Table 1: Parameters for experiments presented here.

The constant exploration strategy [14] was used, population size was 8000. Each explore run (a complete mapping) was followed by one exploit run of the mapper. One experiment consisted of at most 80000 mapping trials. Each mapping trial was terminated when either all blocks were assigned to CLBs, or when 200 actions were executed. All plots shown contain values only for the exploit runs.

Benchmark	#gates	#DFFs	#ABN Nodes
MUX	61	0	61
MODULO12	16	4	20
DK16	135	5	140

Table 2: Benchmarks and their parameters

5.1 ADAPTING TO A SPECIFIC ARCHITECTURE

The principal results of this experiment are shown in Figure 7 and Figure 8 (two figures are shown to demonstrate the performance for examples of combinatorial and sequential circuits). Note that each of these results is from a single run. The performance plotted in these graphs has a direct relation to the performance of the resulting circuit and thus of the mapper, it shows a cumulative reward (see Section 4.1) over one mapping trial. Note that the actual performance plots are a 50 point moving average (as is typical in XCS performance plots), but the best performance plots may be a more realistic performance measures on this task, since rule sets are intended to be extracted, and used offline. A statistical overview of the learning task is shown in Figure 9 and Figure 10, which are averages over 10 learning runs. Note that these results are not subjected to a moving average, since that would tend to cloud the meaning of error bars.

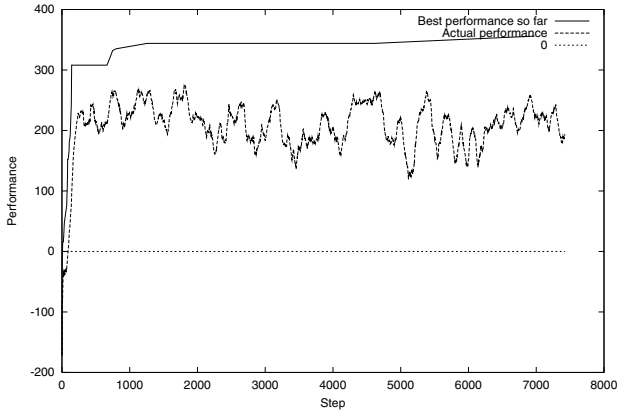


Figure 7: Improvement in performance - combinatorial circuit (MUX). Results from a single run, in training mode.

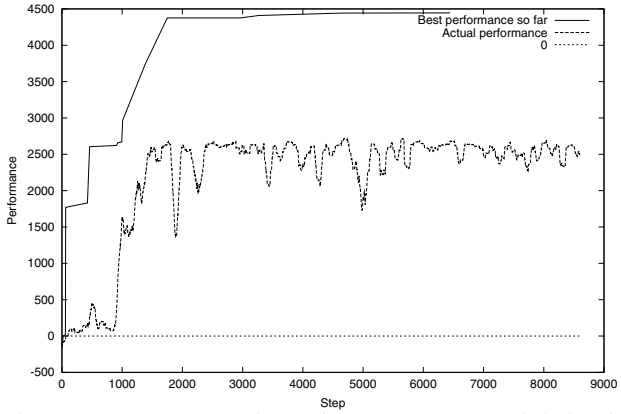


Figure 8: Improvement in performance - sequential circuit (DK16). Results from a single run, in training mode.

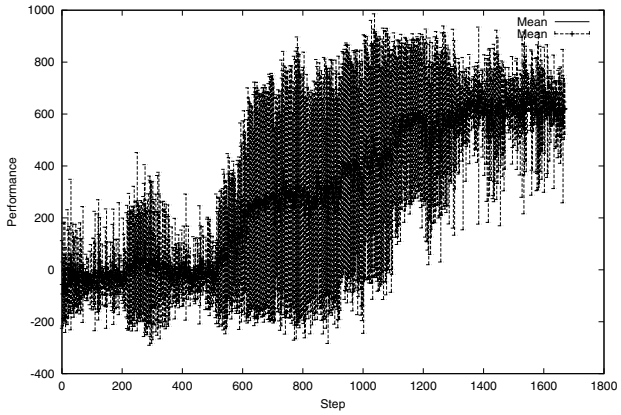


Figure 9: Simple sequential circuit (MODULO12), training mode, 10 independent runs, mean and standard deviation.

At the beginning, the mapper uses randomly generated mapping rules. This corresponds to the initial region with low performance. Then, as more mapping trials are performed, the quality of the mapped circuit, in terms of the performance, improves.

Note the differences between Figure 7 and Figure 8. The initial flat region lasts for more mapping steps in the case of the sequential circuit than for the combinatorial circuit.

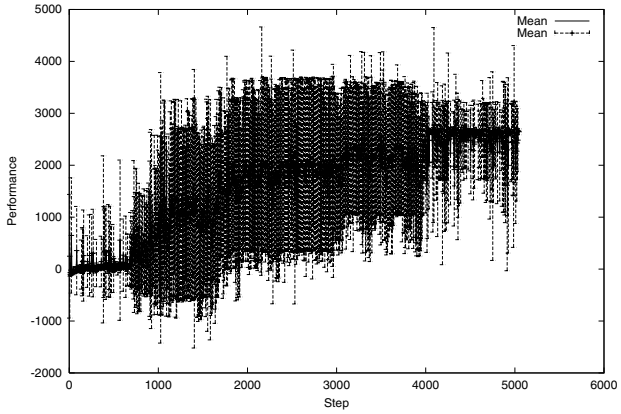


Figure 10: Sequential circuit (DK16), training mode, 10 independent runs, mean and standard deviation.

This is because learning a good mapping policy for sequential circuits is more difficult than for combinatorial circuits. Sequential circuits employ an additional logic element not found in combinatorial circuits (a flip-flop), and the critical path in such circuits is formed by stages of combinatorial circuits connected by flip-flops, which means that changes caused by mapping actions have a more local effect. The differences between Figure 9 and Figure 10 show that simple circuits can be mastered more quickly and efficiently than more complex circuits.

5.2 KNOWLEDGE TRANSFER

In this application, it is desirable that the evolved rule-sets can be shared among different designs, once they are discovered. The results in Figure 11 show that this is feasible for one design. Results in Figure 12 show that rules evolved for a simpler design can be used to improve the performance of the mapper for a more complicated design. Compare these results to the early generations of Figure 8, where no pre-evolved rules were included.

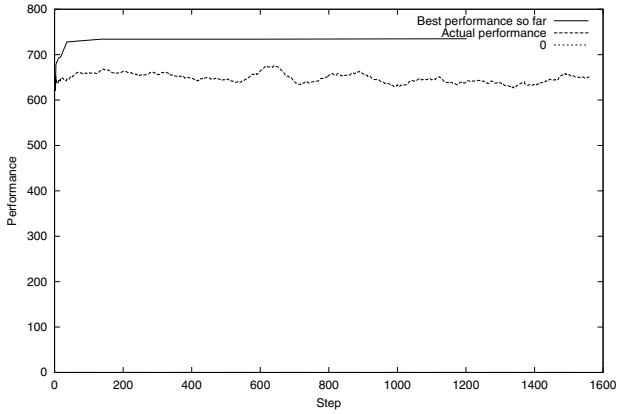


Figure 11: Results of using rules evolved in an experiment that started with an empty rule-set on the same simple sequential circuit. Pre-evolved rules for this experiment were taken from step 8000 of the training run.

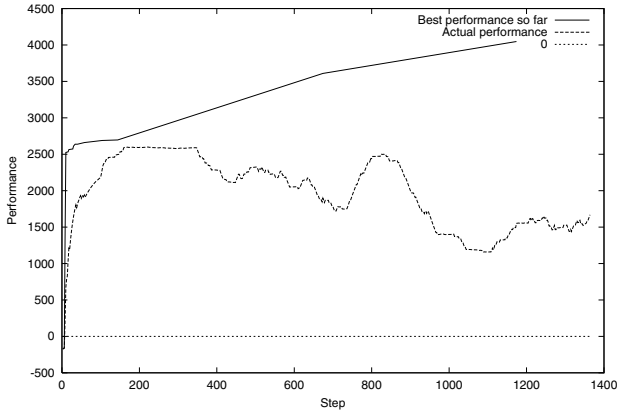


Figure 12: Rules evolved in an experiment that started with an empty rule-set on a smaller circuit – MODULO12 used on a larger sequential circuit (DK16). Pre-evolved rules for this experiment were taken from step 8000 of a training run.

6 DISCUSSION

Despite the relative success shown in these experiments, the task faced by the adaptive mapper is a complex one, and methods of reducing these complexities must be explored. Aspects of the task that may impede XCS include:

- The task is highly sequential, which means that the classifier system has to evolve long chains of rules.
- The mapping task environment is non-Markovian,
- The states in the search space are visited far from uniformly in the learning process, which may cause difficulties in Q-learning [1]. An improvement might be to consider only those states that are important to find a good policy; the problem is they are not known in advance. In general, there may be an exponential number of such states, because they are in tight connection with all possible mappings of the abstract Boolean network.
- The actions used by the classifier system have a different probability of triggering a reward. The reward is most often generated by the *generate CLB* action, but the effect of this action depends on all other actions in the (long) action chain that are not rewarded most of the time. The Q-learning mechanism is intended to ensure that the reward is distributed equally among all actions to reflect their effects, but the previously mentioned complications may severely hamper this effect.

These observations suggest that the actions may be better viewed as a hierarchy, according to the probability of receiving a reward from the environment:

- Lowest level actions: positioning actions - *moveBk, moveFw*,

- Mid-level actions: cluster generation - *assign DFF, expand LUT*,
- Highest level actions: supercluster generation - *generate CLB*

It may also be possible to improve the efficacy of the classifier system approach either by reducing the state space (omitting some information from the sense vector), or by 'hard-wiring' some a priori knowledge in the classifier system structure. The former approach is undesirable, since it would only increase the amount of hidden information in this non-Markov environment. The second approach appears more viable.

In response to these observations, a modification of the structure of the classifier system is introduced in the following section.

7 MULTI-POPULATION CLASSIFIER SYSTEM

The classifier system modifications introduced here are based on the action hierarchy discussed above, and on ideas presented in [11]. Also see [2] and [12].

In typical classifier systems there is only one population of rules (or a set of actions) and the system considers all rules (and the associated actions) at every time step. This approach presents difficulties in the mapping task, since the classifier system has to learn an efficient ordering of actions that can be deduced beforehand.

Simply stated, an optimal sequence of actions would be of the form

1. Repeat the following for some (unknown) number of steps
    a. move from the current position to a neighboring position
    b. decide whether the current position is to be a part of any of the currently generated clusters
2. generate a CLB.

Each high-quality sequence contains *move to a neighbor block* and *assign to/expand a cluster* actions, and is terminated with a *generate CLB* action. After performing a non-positioning action (i.e., cluster formation) at a position, it does not make sense to perform another non-positioning action, because a block can be part of only one cluster. On the other hand, it makes sense to perform a positioning action directly after another positioning action, for the same reason. Given this prerequisite knowledge, one can easily divide the actions the classifier system works with into several sets. One can supply a finite state machine (FSM) that uses the sense vector and a history of past actions to switch among the sets. This results in reducing the information that the classifier system has to discover, because it is contained in the FSM. It also reduces the search space that needs to be sampled by the genetic algorithm, because the action subsets are usually smaller than the original action set, and because the sense vector can be reduced according to the character of the actions in each set. The reward is calculated as if there was only one set of actions, so that the action sets influence each other directly.

To implement this idea in the adaptive mapper, we construct one population of positioning actions, and another population of all other actions (supplemented with a *do nothing* action). The FSM has only two states (each of which corresponds to one of the two populations). The FSM switches states when the last action was a non-positioning action, or the last action was a positioning action and the current block has yet to be included in any cluster.

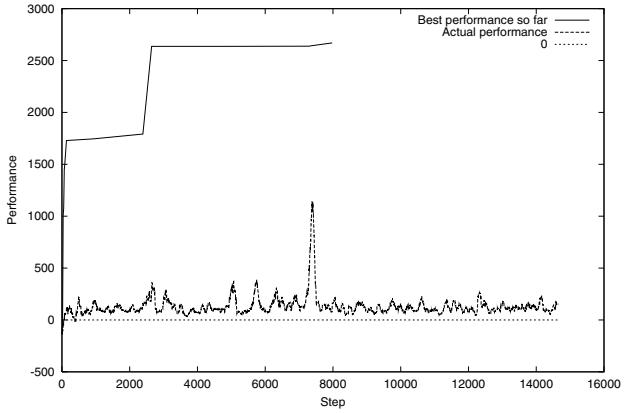


Figure 13: Single population mapper - sequential circuit (DK16), training mode, single run.

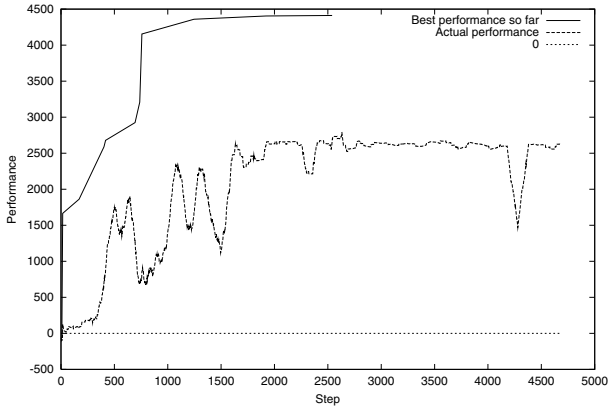


Figure 14: Two population mapper - sequential circuit (DK16), training mode, single run.

The performance of equivalent single- and multi-population mappers is shown in Figure 13 and Figure 14, respectively. Clearly, the two-population mapper performs better than the single-population mapper (note the 'Actual performance' curves rather than the 'Best so far' curves). The results shown are for single runs, to clarify behavior, but they are typical of many runs that we have performed.

8 FINAL COMMENTS AND FUTURE DIRECTIONS

Although this is a work in progress, at least two promising results have been described. First, it has been demonstrated that knowledge gained through XCS training on a small problem can improve performance of the system on a larger problem. Second, a novel approach to hierarchical tasks, using two populations and a problem-specific finite state machine, has been demonstrated to be effective. On the other hand, it is fair to say that any conventional heuristic would perform better than XCS at this moment, mainly because a heuristic was fully adapted to the problem domain by a human programmer. Clearly, given the complexity of this task, more study is needed. In addition to considering additional cases, several points need to be further addressed, including:

- Different methods of credit assignment, particularly epochal schemes, given the episodic nature of this task.
- Consideration of modifications to the classifier system, including the introduction of memory, and examination of ZCS [2], [7].

There are also key concepts in this work that deserve further broader consideration in other applications. These include knowledge transfer from simpler to more complex problems. Another promising area is the use of FSMs and multiple classifier populations to deal with hierarchical tasks. The application presented here used problem-specific information to construct the appropriate FSM, and its interactions with the classifier populations. Such problem-specific design is always good practice in GA-based applications. However, one can also imagine exploring this technique more generally, and allowing the FSM itself to be a subject of adaptation. This remains an interesting area for future investigation.

Acknowledgments

The authors would like to thank Larry Bull for useful and ongoing discussions that are helping to shape this research effort.

References

[1] Barto, A. G., Sutton, R. S. (1998). *Reinforcement learning*. MIT Press.

[2] Barry, A. (2001) A Hierarchical XCS for Long Path Environments. In L. Spector et al. (eds) *GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference.* Morgan Kaufmann, pp 913-920

[3] Bull, L., Hurst, J. (in press). ZCS: Theory and practice. *Evolutionary Computation.*

[4] Cong, J., Ding, Y., Gao, T., Chen, K. (1994). LUT-based FPGA technology mapping under arbitrary net-delay models. *Computers and Graphics.* 18(4).

[5] Danek, M., Muzikar, Z. (1999). Global routing models. In Lysaght, P., Irvine, J., Hartenstein, R. (eds.) *Field-Programmable Logic and Applications: 9th International Workshop, FPL'99*. Springer Verlag.

[6] Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

[7] Lanzi, P. L. (1997). A model of the environment to avoid local learning with XCS in animat problems. *Technical report no. 97.46, Dip. Di Elettronica a Informazione, Politecnico di Milano.*

[8] Murgai, R. , Brayton, R. K., Sangiovanni-Vincentelli, A. (1995). *Logic synthesis for field-programmable gate arrays.* Kluwer.

[9] Servit, M., Muzikar, Z. (1994). Integrated layout synthesis for FPGAs. In Hartenstein, R., Servit, M. (eds.) *Field-Programmable Logic: Architectures, synthesis and applications: 4th International Workshop on Field Programmable Logic and Applications, FPL '94* .Springer Verlag.

[10] Servit, M., Yi, K. (1997). Technology mapping by binate covering. In Luk, W., Cheung, P. Y. K. (eds.), *Field-Programmable Logic and Applications: 7th International Workshop, FPL'97*. Springer Verlag.

[11] Wiering, M., Schmidhuber, J. (1997). HQ-learning. *Adaptive Behavior 6*: (2), MIT Press, pp 219-246.

[12] Wilson, S.W. (1987). Hierarchical Credit Allocation in a Classifier System. In Davis, L. (ed.) *Genetic Algorithms and Simulated Annealing*. pp. 104-115. Pitman.

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

[14] Wilson, S. W. (1996). Explore/exploit strategies in autonomy. In P. Maes, M. Mataric, J. Pollack, J.-A. Meyer and S. W. Wilson (eds.), *From Animals to Animats 4: Proceedings of the Fourth international Conference on Simulation of Adaptive Behaviour*. MIT Press.

[15] Xilinx (2001). Programmable logic data book (Xilinx)

[16] Collaborative Benchmarking Laboratory at North Carolina State University - Web Pages [online] http://www.cbl.ncsu.edu

A Modified Classifier System Compaction Algorithm

Chunsheng Fu
NuTech Solutions, Inc
28 Green St,
Newbury, MA 10951

Lawrence Davis
NuTech Solutions, Inc
28 Green St,
Newbury, MA 01951

Abstract

Although classifier systems have displayed performance levels equaling or exceeding those of other techniques on a variety of benchmark classification problems, they usually solve those problems with a very large number of classifiers. In most cases, a large portion of the final classifier set is unneeded or wrong, with behavior masked by the correctly-functioning rules in the system. Wilson described a post-processing procedure for reducing the number of classifiers in an XCSI classifier system while minimizing the impact of the reduction on the performance level of the system as a whole (Wilson 2001). Wilson's procedure was designed for classifier systems that had been highly trained so that the classifiers were general in nature, and that were always correct in their classifiction of test data. In this paper, we describe some different compaction procedures that can be applied to classifier system sets that are less well-trained, that classify some instances incorrectly, or that contain classifiers that are not fully general.

1 MOTIVATION

XCS classifier systems (Wilson 1995) are competitive with other techniques on real-world classification problems and on benchmark classification problems. XCS's fitness is based on the accuracy of a classifier's payoff prediction. This gives XCS significant improvements on prior classification with respect to prediction accuracy and generality of rules. XCS's capability in both classification and knowledge abstraction makes it unique in solving a variety of real world problems.

One potential benefit of using a classifier system for classification that is frequently mentioned is the possibility that a human might inspect the rules in the system and thereby understand what the system is doing, as compared, for example, with a trained neural network, that contains procedures embedded in a network described by matrices of real-valued numbers. This potential benefit is not fully achieved when the standard approach of training a classifier system to produce hundreds or thousands of classifiers is used, for the following reasons:

- Most of the members of the final set of classifiers do not contribute to the performance of the system as a whole
- Many of those classifiers produced late in the evolutionary process have not been tested, and would degrade performance of the system as a whole, except that more experienced classifiers mask their effects
- Many of those classifiers that are less accurate or less general could be eliminated from the system without impacting performance

For these reasons, when a classifier system is trained, it is likely to contain a majority of macroclassifiers that confuse a human inspecting the system, are inferior in performance to other classifiers in the system, or are wrong but were generated through the evolutionary process and have not yet been eliminated.

Wilson addressed the need for a process that "compacts" a trained set of classifiers by specifying a procedure that could be used on an XCSI system to reduce its size from thousands of classifiers to 20-30, in the examples he considered (Wilson 2001). Wilson's procedure yielded dramatic reductions in classifier system size while resulting in low levels of performance reduction on test sets. Wilson used the Wisconsin Breast Cancer data (Blake 1998) as one of the reference problems on which he conducted his experiments, and we have followed him in the use of this problem in the experiments reported below.

Wilson's procedure works well with highly-trained classifier systems containing accurate and general classifiers. But it cannot be used to reduce the size of less well-trained classifier systems, if they produce classifications on training examples that differ from and example's "true" classification.

In this paper we consider some variant procedures that can be used in these other types of situations.

2 ALGORITHM ANALYSIS

2.1 APPROACH 1

Wilson's compact ruleset algorithm ("CRA") operates on a well-trained XCSI classifier system, and begins after the classifier system has achieved perfect performance. The reader is referred to Wilson 2000 for an explanation of that procedure. The procedures here are heavily inspired by Wilson's approach, but have some different features owing to the need to handle classifier systems that do not display 100% performance after training.

The procedure we began with is closest to Wilson's approach, although it differs in several respects. We call it Approach 1. It proceeds as follows.

Step 1: Beginning with the first classifier in the list, eliminate that classifier from the system and determine the level of performance of the resulting system on the training data. If the level of performance is worse or unchanged, delete the classifier from the system. Terminate step 1 as soon as a classifier is found whose deletion reduces the level of performance of the system as a whole. The remaining set of classifiers, including the one whose deletion reduces performance, is used as the input to step 2.

Step 2: Continuing along the list of classifiers, now eliminate each classifier, in order, and consider the performance of the remaining members of the classifier set. If the level of performance is reduced on deletion, retain this classifier. However, do not use this classifier in the subsequent tests in this step. The set of retained classifiers—those that caused performance reductions in this step—is used as the input to step 3.

Step 3: Construct a final set of classifiers (initially empty), a reference set of instances (initially equal to the training data set) and a set of trial classifiers (initially equal to the output of step 2. Repeat the following procedure until the reference set is empty or no classifier in the trial classifier set matches any member of the reference set: Determine how many members of the reference data set each member of the current trial classifier set matches; move the classifier matching the highest number of members of the reference data set to the final set; and delete the instances that it matches from the reference set. Step 3 could create a set of classifiers that match all the examples in the training set, while preferring general classifiers over specific ones. The final set of classifiers produced in this way is the output of our Approach 1 to classifier system compaction.

2.2 COMMENTS ON APPROACH 1

Approach 1 also results in dramatic levels of compaction on the Wisconsin Breast Cancer database problem. In Wilson's paper, Wilson uses classifier systems trained by presentation of 2,000,000 instances, and we suspected that an approach somewhat inspired by his might be sensitive to training levels. As we will show, a high level of training is necessary for best performance of Approach 1. The Wisconsin Breast Cancer problem can be solved to an equally high level of performance after the presentation of 40,000-80,000 instances. One question we consider below is how well Approach 1 works when training levels are in that range.

It is important to note that any rule compaction procedure has two metrics of interest: performance on the training data set (the data used to train the system), and performance on the test data set (a set of instances drawn from the same distribution that were not used in training). The more important metric is the second, and it is the second that we will primarily consider in this paper. It is well-known that classification systems working on data sets with "noisy" or inappropriate classifications can degrade performance on test data if they are overtrained—trained for extremely long periods of time, or trained so that they have the ability to "memorize" anomalous instances in the training set, resulting in reduced levels of generalization on the test set. There is a danger that a procedure requiring very high levels of training, while resulting in high performance on the training set, will actually degrade performance on the test set. We have shown that this can be the case for the Wisconsin Breast Cancer database (Fu 2001). Thus, there may be a practical as well as a performance-related need for rule compaction procedures that work well on classifier systems that are not highly trained. In addition, for a complicated real-world problem (or even a synthetic one such as the 70-multiplexer problem) there may not be enough time available to fully train a classifier system.

With regard to performance on the training set, it is worth noting that Approach 1 maintains performance levels explicitly in steps 1 and 2—no classifier is deleted whose performance reduces the level of performance of the classifier system as a whole. In step 3, performance is not used as a criterion. Instead, coverage of the training set is used. As we will show later, this causes a significant degradation in terms of prediction accuracy.

Step 3 of Approach 1 has some advantages over a performance-related criterion. The final set of classifiers produced by Approach 1 can be smaller than our performance-related criteria, as we will see.

In the remainder of this paper, our version of XCSI uses the following parameter values throughout all experiments: Population size is 3200, learning rate is 0.25, α is 0.1, Error threshold (e0) is 1, γ is 5, GAThreshold is 48, Crossover Probability is 0.8, Mutation Probability is 0.04, Deletion Threshold Experience is 50, Deletion Threshold Fitness is 0.1, Subsumption Threshold Experience is 100, Minimum Number of Actions in match set is 1, Fitness Updating Coefficient is 0.1, Error Updating Coefficient is 0.25, CoverRange ([0,r0]) is 6, Mutation Range ([1,m0]) is 2, and the reward/penalty values are 100/-100

3 RESULTS AND DISCUSSION

The Wisconsin Breast Cancer database, donated by Prof. Olvi Mangasarian, is a database of real-world data collected by Dr. William H. Wolberg to serve as a test case for classification data mining systems (Blake 1998). There are 699 records in the database, and each contains values for 9 attributes. The attribute values are integers, and each ranges between 1 and 10. The attributes have to do with properties of tissue samples, such as: clump thickness, uniformity of cell size, etc. Each record is classified as either benign or malignant. The task of a data mining system on this database is to use the attributes of records whose classification is known ("training records") to learn to predict whether an unseen case (a "test record") is benign or malignant. In other words, the task is to discover patterns and regularities in the data that allow reliable prediction of an unseen record's classification. The measure of performance of a system on this task is the system's accuracy at predicting records that it has not seen during training. It should be noted that a small number (16) of the records in the WBC database have some missing attributes. Our version of XCSI followed the procedure in Wilson's version by regarding a missing attribute as matched by any classifier.

Table 1: Performance of Approach 1 on different classifier sets

Training instances	**5K**	**50K**	**200K**	**1000K**
Initial P	0.9282	0.9207	0.9422	0.9544
Step 1 P	0.9282	0.9137	0.9422	0.9572
Step 2 P	0.9064	0.9062	0.9356	0.9529
Step 3 P	0.6982	0.8919	0.8790	0.9072
Size of CR	23.5	24.0	15.5	14.5

(P means performance; CR means compact rule set)

Approach 1 produces some reduction of performance level on both the training set and the test set as shown in Table 1 and Table 2.

In Table 1, we show the results of using Approach 1 on a run of tenfold stratification of the Wisconsin Breast Cancer (WBC) database, using our implementation of XCSI (Fu 2001). Classifiers (actually, *macroclassifiers*, many of which have numerosity greater than 1) are ordered by numerosity throughout the experiments reported in this paper. Results are presented for four levels of training of the classifier system: 5,000, 50,000, 200,000 and 1,000,000 trials.

The table shows the level of performance of the output of each of the three steps of the compaction procedure on the test data. Wilson's statement that his compaction procedure works best on highly trained classifier systems is borne out here for Approach 1. The highest levels of performance on test data, after compaction, are achieved when the compaction procedure is carried out on classifier systems that have seen the highest number of training examples—much higher numbers than those required to train the system to its optimal level of performance.

Let us consider some points related to the level of performance reduction in Table 1. As a reference, Wilson's application of XCSI without rule compaction to the Wisconsin Breast Cancer database produced results (95.5% accuracy on unseen instances, using tenfold cross-validation) that were better than any previously published results, which were in the range of 94-95% accuracy. A rough characterization of the levels of accuracy on this problem is that 93% accuracy could be achieved by nearly any technique applied to the data—decision trees and neural networks easily achieved this level of accuracy. Prior to Wilson's work on XCSI, 94.5% was state of the art, and anything higher was new ground.

Considering these levels of performance, we see that compaction of the data using Approach 1 reduces the performance of the system in each case well below the level achievable by most of the rival techniques. This might be a problem for classifier system acceptance in, for instance, the commercial arena: if compaction of a set of classifiers to a human-comprehensible size results in performance levels well below those of competing techniques, then classifier systems may not be preferred to decision trees, for example, whose classification strategy is also human-readable, but has higher performance when pruned to comparable levels of simplification.

For this reason, we did extensive experiments on Approach 1, monitoring each of its three reduction steps with regard to performance on the training data. Table 2 displays the initial prediction performance over training data, the initial rule set size, size of the compact rule set after each step, and the final compact rule set's performance on the training data.

Table 2: Performance of Approach 1 on training set during compaction

Training instances	5K	50K	200K
Initial P	0.9730	0.9952	0.9984
Initial Size	1859.0	1863.5	1381.5
CR size after S1	1188.5	585.5	252.0
CR size after S2	80.0	52.0	38.5
CR size after S3	23.5	24.0	15.5
Final Performance	0.7989	0.9793	0.9499

(P means performance; CR means compact rule set; S_i means step i)

As shown in Table 2, the size of the compact rule set decreases if the initial classifiers are trained over more instances. The more training, the less classifiers are needed to represent the system. Also, we note that the more training, the more classifiers are removed by the first and second reduction steps. Finally, note that performance was significantly degraded even over the training data. Since the first two steps of Approach 1 prevent performance degradation over the training data, the performance loss results from step 3. Thus, we considered modification to step 3 in our work on compaction algorithms. We experimented with two variations on Approach 1, which we describe below.

Two modifications to Approach 1

The first variation we implemented was incremental deletion of classifiers. Note that Wilson's original algorithm works at the macroclassifier level—each macroclassifier with numerosity greater than 1 really represents multiple classifiers, and Approach 1 follows him in this. We hypothesized that deleting microclassifiers one at a time, and testing the result on subsequent performance, might yield better "balanced" sets of classifiers. The all-or-nothing approach might produce performance degradations related to the high numerosity of the surviving macroclassifiers, or so we thought.

We applied our incremental deletion procedure to step 2 of Approach 1, yielding what we called Approach 2. We didn't consider step 1, since the result of both approaches to deletion is the same in step 1. In step 2, deleting microclassifiers has the potential to "reweight" the classifier system, yielding more appropriate strengths on the relative recommendations made by the system, after deletion of classifiers whose weight was important to the system's performance.

As we show below, microclassifier deletion does not improve the performance of the system after compaction, and it appears to slightly degrade performance over macroclassifier deletion on the WBC problem. This is very likely because of the "reweighting" effect. Since step 2 only considers the performance of M_i over M_{i-1} (not all classifiers), the reweighting may result in a problem for a slightly-favored action (Fu 2001). We believe that further study of the incremental deletion is necessary, although our experiments did not show that it is useful for the compaction approaches we tested.

The second variation we studied was a different procedure for step 3 of Approach 1, yielding Approach 3. Table 1 shows that the most significant reductions in performance of the compaction algorithm occur at step 3, where performance is not considered when the final classifier set is built. We experimented with a variant version of step 3 that was more in the spirit of steps 1 and 2. The step can be described as following. For a macrostate ordered by numerosity or experience in increasing order, delete the last classifier and check the performance of the remaining classifiers. If the performance is degraded, then the just deleted classifier is reinserted into the head of the classifier list, and so is retained and used in subsequent tests. Repeat the process until every macroclassifier has been tried by this kind of deletion.

Table 3: Performance of Approach 3 on test data

Training instances	5K	50K	200K
Initial P	0.9282	0.9282	0.9422
Step 1 P	0.9282	0.9137	0.9422
Step 2 P	0.9064	0.8921	0.9356
Step 3 P	0.8915	0.8772	0.9217
CR size	41.5	26.0	24.0

(P means performance; CR means compact rule set;)

Table 4: Performance of Approach 3 on the training data

Training instances	5K	50K	200K
Initial P	0.9730	0.9952	0.9984
Initial Size	1859	1863.5	1381.5
CR size after S1	1188.5	585.5	252.0
CR size after S2	80.0	52.5	38.5
CR size after S3	41.5	26.0	24.0
Final Performance	0.9738	0.9960	0.9992

(P means performance; CR means compact rule set; S_i means step i)

Table 3 shows the performance levels of Approach 3 on identical classifier systems trained on the Wisconsin Breast Cancer database for 5,000, 50,000, and 200,000 instances. If we contrast the data in Table 3 with that in Table 1, and if we note that both tables were constructed based on the compaction of identical initial classifier systems, we can see several differences between the behavior of Approach 1 and Approach 3.

The first is that Approach 1 yields smaller sets of classifiers—the output of step 3 is 15.5 classifiers versus 24.0, in the 200,000 case. A second difference is in the levels of performance. After step 2, the microclassifier

deletion technique shows slightly worse performance on the 50,000 case. However, after step 3, the performance-based compaction technique shows substantially higher levels of performance on the test data. We see 89% versus 70% and 92% versus 88% for the 5,000 case and the 200,000 case, although in the 50,000 case, the original procedure does better, with 89% versus 88%. (Our later experiments showed in Table 6 that the ~1% loss is created during step 2.)

As shown in Table 2 and Table 4, both original step2 and the modified step2 reduced the same number of classifiers. To summarize, we can see that lower training levels produce compact rule sets with lower levels of performance for both versions of the compaction algorithm, but performance loss is much greater for Approach 1. We see that Approach 1, however, produces rule sets that are smaller than those produced by Approach 3, and so some tradeoffs are possible when selecting compaction algorithms.

We wished to learn more about the effects of the variant versions of the three steps. To do this, we used highly-trained sets of classifiers (one million instances of training) as input to three versions of the compaction algorithm: Approach 1, Approach 2, and Approach 3. Table 5 shows the results of this study.

Table 5: Comparison of Approach 1 with Approach 2 and Approach 3 on a classifier system trained over 1,000,000 instances

Alg Names	S1S2S3	S1S2mS3	S1S2mS3m
Initial P	0.9544	same	same
Step 1 P	0.9572	same	same
Step 2 P	0.9529	0.9515	0.9515
Step 3 P	0.9072	0.9015	0.9343
Size of CR	14.5	14.5	20.9

(P means performance; CR means compact rule set; Si means Approach 1's step i; Sim means our modified procedure for step i; Si P means step i's performance)

There are several points to note concerning the data in Table 5. First, we see again that the size of the final set is larger when the performance-based version of step 3 is used, as in Approach 3. Second, we see that Approach 3 produces higher levels of performance on the test data. Third, we see that Approach 2 yields worse performance for the original step 3.

Our experimental results suggest a reduction procedure using Approach 3, unless rule set size is an important consideration. That is, we recommend in general using step 1 and step 2 of Approach 1 and our modified step 3. We implemented this reduction procedure and the results, shown in Table 6, support this recommendation.

Table 6: Performance of suggested CRA (S1S2S3m) on different classifier sets

Training instances	5K	50K	200K
Step 2 P	0.9064	0.9062	0.9356
Step 3 P	0.8915	0.8990	0.9217
CR size	41.5	26.0	24.0

(P means performance; CR means compact rule set;)

4 CONCLUSIONS

The XCS family of classifier systems already competes well with other approaches to classification. If it is to compete on problems requiring compact, human-readable solutions, then effective classifier system rule compaction procedures will be needed. In this paper we have discussed three approaches to rule set compaction that were inspired by Wilson's work, but that differ so that they can be applied to the compaction of classifier systems that do not have high levels of generalization or perfect accuracy on all test set examples. Approaches 2 and 3 yield classifier systems of compact size on the Wisconsin Breast Cancer database. They also yield higher levels of performance than Approach 1 on unseen data, and they yield lower numbers of unmatched instances. They also yield reduced sets of larger size than Approach 1.

To conclude, we know that uncompacted classifier systems are already competitive with all other classification techniques with regard to performance level, but they are not compact and are not human-comprehensible. We hope that Wilson's paper and this one will stimulate further work in classifier system compaction, in order to increase the range of real-world situations in which classifier systems are indeed the algorithm of choice for solution of classification problems, and to realize the possibility that the classifier system approach produces both high-performance results and compact sets of high-quality rules.

References

Blake, C. and C. Merz (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html

Butz, M. V. and Wilson, S. W. (2001). An Algorithmic Description of XCS. In Lanzi, P. L. , Stolzmann, W., and S. W. Wilson (Eds.) *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS-2000).* Springer-Verlag. Or see http://prediction-dynamics.com

Fu, C. S, Wilson, S. W. and Lawrence, D (2001). Studies of the XCSI classifier system on a datamining problem. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), page 95.* Morgan Kaufmann: San Francisco, CA, 2001.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2), 149-175

Wilson, S. W. (2000). Mining Oblique Data with XCS. In: Technical Report No. 2000028, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign

A Comparison between ATNoSFERES and XCSM

Samuel Landau* **Sébastien Picault***
*Laboratoire d'Informatique de Paris 6
8 rue du Capitaine Scott
75 015 Paris

Olivier Sigaud*
**Dassault Aviation
DGT/DPR/ESA
78, Quai Marcel Dassault
92552 St-Cloud Cedex

Pierre Gérard*,**

Abstract

In this paper we present ATNoSFERES, a new framework based on an indirect encoding Genetic Algorithm which builds finite-state automata controllers able to deal with perceptual aliasing. We compare it with XCSM, a memory-based extension of the most studied Learning Classifier System, XCS, through a benchmark experiment. We then discuss the assets and drawbacks of ATNoSFERES in the context of that comparison.

Keywords

Evolutionary Algorithms, Learning Classifier Systems, perceptual aliazing, Augmented Transition Networks

1 Introduction

Most Learning Classifier Systems (LCS) (5) are used to tackle problems where situated and adaptive agents are involved in a sensori-motor loop with their environment. Such agents perceive situations through their sensors as vectors of several attributes, each representing a perceived feature. The task of the agents is to *learn* the optimal policy – *i.e.* which action to perform in every situation, in order to fulfill their goals the best way they can. Like in the general *Reinforcement Learning* (RL) framework (17), the goals of LCS are defined by scalar rewards provided by the environment. The policy is defined by a set of rules – or classifiers – specifying which action to choose according to *conditions* about the perceived situations.

In real world environments, it may happen that agents perceive the same situation in several different locations, some requiring different optimal actions, giving rise to *perceptual aliazing* problems. In such cases, the environment is said *non-Markov*, and agents cannot perform optimally if their decision at a given time step only depends on their perceptions at the same time step. Though they are more often used to solve Markov problems, there are several attempts to apply LCS to non-Markov problems, like (18, 10) for instance.

Within this framework, explicit internal states were added to the classical (condition, action) pair of the classifiers (11, 10, 20). These internal states provide additional information to choose the optimal action when the problem is non-Markov. The problem of properly setting the classifiers, and setting the internal states in particular, is devoted to *Genetic Algorithms* (GA).

In this paper, we will compare LCS to "ATNoSFERES", a new system that also uses GA to automatically design the behavior of agents facing problems in which they perceive situations as vectors of attributes, and have to select actions in order to fulfill their goals, in non-Markov environments. In ATNoSFERES, the goals are defined thanks to a *fitness* measure.

In the first section, we present the features and properties of the ATNoSFERES model (9, 15). It relies upon oriented, labeled graphs (§ 2.1) for describing the behavior and the action selection procedure. The specificity of the model consists in building this graph from a bitstring (§ 2.2) that can be handled exactly like any other bitstring of a Genetic Algorithm, with additional operators. Then we show that the graph-based representation is formally very similar to LCS representations, and, in particular, to XCSM (§ 3.2); thus we compare both approaches through classical experiments (§4). As a result of this comparison, we discuss the assets and drawbacks of both representations according to different criteria (§5). Finally, we conclude by stating what should be added to ATNoSFERES so as to improve it further where the comparison is not

in its favor.

2 Description of ATNoSFERES

2.1 Graph-based expression of behaviors

The architecture provided by our model involves an "Augmented Transition Networks" (ATN)-like graph (21) which is basically an oriented, labeled graph with a Start (or initial) node and an End (or final) node (see figure 5). Nodes represent states and edges represent transitions of an automaton.

Such graphs have already been used for describing the behavior of agents (9). The labels on edges specify a set of conditions (*e.g.* `c1 c3 ?`) that have to be fulfilled to enable the edge, and in a sequence of actions (*e.g.* `a5 a2 a4!`) that are performed when the edge is chosen. We use those graphs as follows:

- At the beginning (when the agent is initialized), the agent is in the *Start* node (S).
- At each time step, the agent crosses an edge:
  1. It computes the set of eligible edges among those starting from the current node. An edge is eligible when either it has no condition label or all the conditions on its label are simultaneously true.
  2. If the set is empty, then an action is chosen randomly; else an edge is randomly chosen in the set.
  3. The edge occurs by performing the actions on the label of the current edge. When the action part of the label is empty, an action is chosen randomly.
  4. The new current node becomes the destination of the edge.
- The agent stops when it is in the *End* node (E).

Note that most of behavioral structures involved in classical evolutionary approaches, *e.g.* program trees in Genetic Programming (7), are entirely interpreted at each time step to determine the actions to perform. It is not the case in our approach which relies on internal nodes. An example of the perception-action cycle performed during each time step is given further on figure 3.

2.2 The graph-building process

The behavioral graph is built from an hereditary substrate, by adding nodes and edges to a basic structure containing only the *Start* and *End* nodes.

There are many different evolutionary techniques to automatically design structures such as finite-state machines (2), neural networks (22) or program trees (7). Very roughly, we can sketch an opposition between, on the one hand, approaches that use the genotype as an encoding of a set of parameters (like Genetic Algorithms (5, 1, 3) or Evolution Stategies (16)) and, on the other hand, approaches that use the genotype as a structure producing the phenotype (such as Genetic Programming (7, 14), Evolutionary Programming (2), L-systems (12), developmental program trees (6, 4, 13)...).

In the ATNoSFERES model, we try to conciliate advantages from both kind of approaches: on the one hand, since the behavioral phenotype is produced by the interpretation of a graph, we want it to be of any complexity; on the other hand, we use a fine-grain genotype (a bitstring) to produce it, in order to allow a gradual exploration of the solution space through "blind" genetic operators.

Therefore, we follow a two-step process (see figure 1) [1]:

1. The bitstring (genotype) is translated into a sequence of tokens.
2. The tokens are interpreted as instructions of a robust programming language, dedicated to graph building.

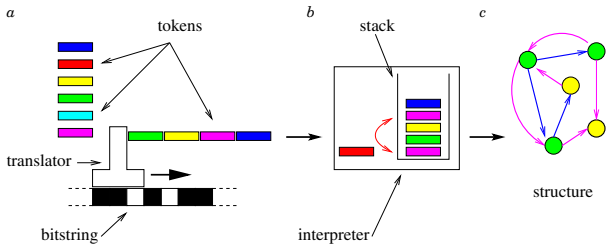


Figure 1: The principles of the genetic expression we use to produce the behavioral graph from the bitstring genotype. The string is first decoded into tokens (a), which are interpreted in a second step as instructions (b) to create nodes, edges, and labels (c).

2.2.1 Translation

Translation is a simple process that reads the bitstring genotype and decodes it into a sequence of *tokens* (symbols). It uses therefore a *genetic code*, i.e. a function $\mathcal{G} : \{0,1\}^n \longrightarrow \mathcal{T}$ $(|\mathcal{T}| \leq 2^n)$ where $\mathcal{T}$ is the set

[1] More details about those mechanisms and the nature of the tokens are provided in (9, 15)

of possible tokens (the different roles of which will be described in the next paragraph). Depending on the number of available tokens, the genetic code might be more or less redundant. Binary substrings of size n (decoded into a token each) are called "codons".

2.2.2 Interpretation

Tokens are instructions of the ATNoSFERES graph-building language. They operate on a stack in which data tokens or parts of the future graph are stored. All tokens fall into the following categories:

- condition or action tokens, which only put data in the stack, which will be used to label edges between nodes;
- node creation or node connection tokens (the latter use nodes and action/condition tokens already in the stack);
- stack manipulation tokens (swap, copy...) which have an action upon the stack containing nodes and action/condition tokens.

In order to cope with a "blind" evolutionary process (i.e. based on random mutations on a fine-grain genotype), the graph built has to be robust to mutations (15). For instance, the replacement of a token by another, or its deletion, should only have a *local impact*, rather than transforming the whole graph.

If an instruction cannot be executed successfully, it is simply ignored; for the same reasons, when all tokens have been interpreted, the graph is made consistent, *e.g.* by linking some nodes to Start/End nodes. Any sequence of tokens is meaningful, thus the graph-building language is robust to variations affecting the genotype (there is no specific syntactical nor semantical constraint on the genetic operators).

2.3 Integration into an evolutionary framework

In this paper, the ATNoSFERES model has been applied to produce agents behaviors within an evolutionary algorithm.

Therefore, each agent has a bitstring genotype from which it can produce a graph (the genetic code depends on the perception abilities of the agent and on the actions it can perform). The fitness of each agent is computed by evaluating its behavior in an environment. Then individuals are selected depending on their fitness and bred to produce offspring.

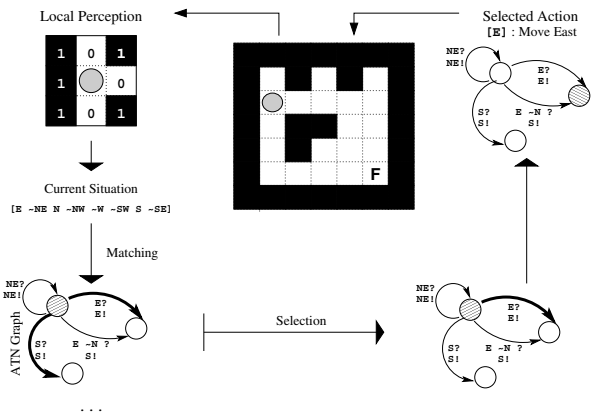


Figure 2: In this example, the agent, located in a cell of the maze, perceives the presence/absence of blocks in each of the eight surrounding cells. It has to decide whether to try to move towards one of the eight adjacent cells. From its current location, the agent perceives `[E ¬NE N ¬NW ¬W ¬SW S ¬SE]` (token `E` is true when the east cell is empty). From the current state (node) of its graph, two edges (in bold) are eligible, since the condition part of their label match the perceptions. One is randomly selected, then its action part (move East) is performed and the current state is updated.

The genotype of the offspring is produced by a classical crossover operation between the genotypes of the parents. Additionally, we use two different mutation strategies to introduce variations into the genotype of new individuals: classical bit-flipping mutations, and random insertions or deletions of one codon. This modifies the sequence of tokens that will be produced by translation, so that the complexity of the graph itself may change. Nodes or edges can in fact be added or removed by the evolutionary process, as can condition/action labels.

3 Learning Classifier Systems

As explained in the introduction, the problems tackled by LCS are characterized by the fact that situations are defined by several attributes representing perceivable properties of the environment. A LCS has to learn classifiers, which define the behavior of the system as shown in figure 3. Within the LCS framework, the use of *don't care* symbols "`#`" in the condition parts of the classifiers results in generalization, since *don't care* symbols make it possible to use a single description to describe several situations. Indeed, a *don't care* symbol *matches* any particular value of the considered

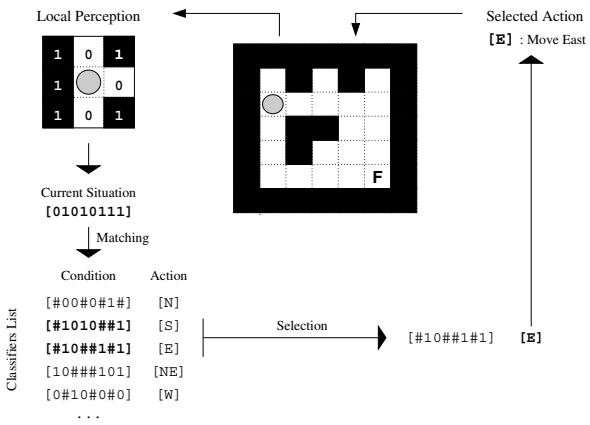


Figure 3: The agent perceives the presence/absence (resp. 1/0) of blocks in each of the eight surrounding cells (considered clockwise, starting with the north cell). Thus from its current location, the agent perceives `[01010111]`. Within the list of classifiers characterizing it, the LCS first selects those matching the current situation. Then, it selects one of the matching classifiers and the corresponding action is performed.

attribute.

The main issue with generalization is to organize conditions and actions so that the *don't care* symbols are well placed. To do so, LCS usually call upon a GA.

In the *Pittsburg* style, the GA evolves a population of LCS with their whole lists of classifiers. The lists of classifiers are combined thanks to crossover operators and modified with mutations. The LCS are evaluated according to a fitness measure and the more efficient ones – with respect to the fitness – are kept. Thus, like in the ATNoSFERES model, a Pittsburg style LCS evolves a population of controllers.

On the contrary, in the *Michigan* style, the GA evolves a population of classifiers within the list of classifiers of a single agent. Here, this is the classifiers which are combined and modified. A fitness is associated to each classifier and the best ones are kept. Thus Michigan style LCS use GA to perform online learning: the classifiers are improved during the life time of the agent. Usually, such LCS rely on utility functions that depend on scalar rewards given by the environment, as defined in the RL framework (17).

In most of the early LCS (5), the fitness was defined directly according to the utility associated to the classifier. After having defined a very simple LCS called ZCS in (19), Wilson found much more efficient to define the fitness according to the accuracy of the utility prediction. Its system, XCS (20), is now the most widely used LCS to solve Markov problems.

3.1 XCSM

Dealing with simple `Condition-Action` classifiers does not endow an agent with the ability to behave optimally in perceptually aliazed problems. In this kind of problems, it may happen that the current perception does not provide enough information to always choose the optimal action: as soon as the agent perceives the same situation in different states, it will choose the same action though this action may be inappropriate in some of these states (see figure 4).

For such problems, it is necessary to introduce internal states in the LCS. Tomlinson and Bull (18) proposed a way to probalistically link classifiers in order to bridge aliazed situations. Lanzi (10) proposed XCSM, where M stands for Memory, as an extension of XCS with explicit internal states. XCSM manages an internal memory register composed of several bits that explicitely represent the internal state of the LCS. Therefore, a classifier contains four parts (*cf.* table 1) an external condition about the situation, an internal condition about the internal state, an external action to perform in the environment and an internal action that may modify the internal state.

The internal condition and the internal action contain as many attributes as there are bits in the memory register. In order to be selected by the LCS, a classifier has to match with both external and internal conditions. When it is selected, the LCS performs the corresponding action in the environment and modifies the internal state if the internal action is not composed only of *don't change* symbols "#". When a classifier is fired, a *don't care* symbol in the internal action results in letting the corresponding bit in the memory register at its value before applying the classifier. As XCS, XCSM draws benefits from generalization in the external condition, but also in the internal condition and the internal action.

The memory register provides XCSM with more than just the environmental perceptions. It permits to deal with perceptual aliazing by adding information from the past experience of the agent.

3.2 Formal relations between ATNoSFERES and Learning Classifier Systems

An ATN such as those evolved by ATNoSFERES can be translated into a list of classifiers, whether they have been obtained through a Michigan or a Pittsburgh style process. The nodes of the ATN play the

role of internal states and permit ATNoSFERES to deal with perceptual aliazing. The edges of the ATN carry several informations which can be translated in a rule-based formalism: the source and destination nodes of the edge can be respectively represented by an internal condition and an internal action; the conditions associated to the edges correspond to the external conditions of the classifiers; the actions associated to the edges correspond to the external actions of the classifiers.

It is clear in our example that an important difference between both formalisms is due to the possibility to perform *a sequence of actions* (such as `a3`–`a5`) as a consequence of matching conditions. We restricted this feature to a single action in the experiments described below (§ 4.3).

There are two other differences, that have been kept in our experiments:

- When the action part of the edge label is empty (represented by a `#` on the graphs), an action is randomly chosen among possible ones. We represent it by a classifier containing only # in the LCS-like formalism. The consequences of that feature will be discussed in §5.
- In XCSM, the "internal state" is regarded as an extension, while it is an inherent feature of the graph-based approach. Hence XCSM may have general rules that match in any situation (whatever the internal state can be, i.e. #).

4 Experiments

4.1 The perceptual aliazing problem

In some environments (like `Maze10` on figure 4), some states may induce identical perceptions by the agent, though different actions must be performed. This defines the "perceptual aliazing" issue that is frequently encountered in real-world environments.

We have compared the nature of the results that have been obtained through Evolution to those produced by a LCS like XCSM (10) in the `Maze10` environment.

4.2 Experimental setup

We tried to reproduce an experimental setup close to that used in Lanzi (10) with the `Maze10` environment, with regards to the specificities of our model.

The agents used for the experiments are able to perceive the presence/absence of blocks in the eight adjacent cells of the grid. They can move in those adjacent cells (the move will be effective when the cell is empty or contains food). Thus the genetic code includes 16 condition and 8 action tokens. In order to encode 24 condition-action tokens together with 7 stack manipulation and 4 node creation/connection tokens, we need at least 6 bits to define a token ($2^6 = 64$ tokens, which means that some tokens are encoded twice).

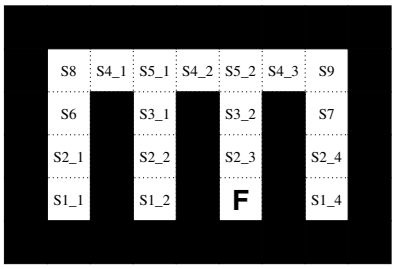


Figure 4: The Maze10 environment. **F** represents the goal to reach (food) from any cell of the maze; a few cells are unambiguous (S_i) but in the other ones the same perceptual situations may require either similar actions or different ones (*e.g.* go north in $S_{2\_\{1,2,4\}}$ but go south in S_{2_3})

Each experiment involves the following steps:

1. Initialize the population with $N = 300$ agents with random bitstrings.
2. For each generation, build the graph of each agent and evaluate it in the environment.
3. Select the individuals with higher fitness (namely, 20 % of the population) and produce new ones by crossing over the parents. The system performs probabilistic mutations and insertions or deletions of codons on the bitstring of the offspring.
4. Iterate the process with the new generation.

In order to evaluate the individuals, they are put into the environment, starting on any blank cell in the grid, and they have to find the food within a limited amount of time (20 time steps). The agent can perform only one action per time step; when this action is incompatible with the environment (*e.g.* go towards a wall), it is simply discarded (the agent loses one time step). Its fitness for each run is: $F = D - K + B + 2 * R$ (F: fitness for the run; D: number of blank cells that have been discovered during the run; K: time steps spent on already known cells; B: bonus when the food is found (30 points); R: remaining time if the food has been found within the time limit ($R < 19$)). It was designed to advantage exploring agents (see D and K)

that reach quicker the food (thanks to the R coefficient, remaining time steps are more rewarding than the discovering of any more new cells). Since there is no reinforcement learning during the run, the fitness *has* to provide delayed information to measure the quality of the behavior. Each agent is evaluated 4 times starting on each empty cell, then its total fitness is the sum of the fitnesses computed for each run. In the optimal case, the fitness is 4500.

The experiments reported here were carried out on various initial genotype sizes, from 300 to 540 bits. The original population genotype sizes change during evolution. Each experiment has been bounded by 10,000 generations, which in most cases is sufficient to reach high enough fitness values.

4.3 Results

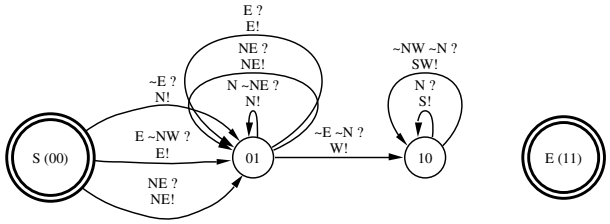


Figure 5: Graph of the best individual in a representative experiment

A representative example is reported on figure 6, which shows the best and average fitness values.

Table 1: A LCS-like representation of the graph on figure 5. EC: external conditions, IC: internal conditions, EA: external actions, IA: internal actions

EC								IC	EA	IA
E	NE	N	NW	W	SW	S	SE			
1	#	#	#	#	#	#	#	00	N	01
0	#	#	1	#	#	#	#	00	E	01
#	0	#	#	#	#	#	#	00	NE	01
#	1	0	#	#	#	#	#	01	N	##
#	0	#	#	#	#	#	#	01	NE	##
0	#	#	#	#	#	#	#	01	E	##
1	#	1	#	#	#	#	#	01	W	10
#	0	#	#	#	#	#	#	10	S	##
#	#	1	1	#	#	#	#	10	SW	##

Figure 5 presents a behavioral graph obtained by the best individual in a representative experiment. It has also been represented in a LCS-like formalism (table 1).

The agent whose graph is described in figure 5 has the following behavior: from any vertical corridor, it first reaches horizontal corridor, then the NE corner, and finally goes straight to the food. This is a nearly optimal solution. The graph presented in figure 5 shows that a nearly optimal behavior can be obtained. Especially, there are clear distinctions between the bottom of vertical corridors (N ¬NE identifies cells $S_{\{1,2\}\_n}$), the top of vertical corridors (NE $\rightarrow S_6, S_7, S_{3\_n}$), the horizontal corridor (E $\rightarrow S_8, S_{\{4,5\}\_n}$) and the crucial NE corner (S_9 is identified by ¬E ¬N ¬NW).

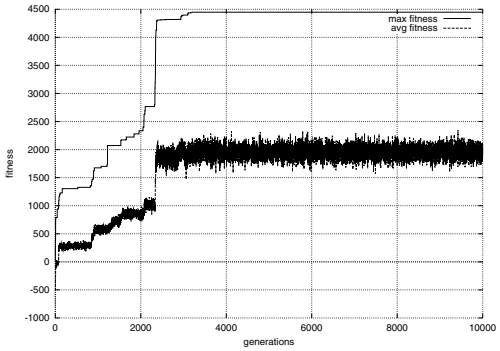


Figure 6: Best and average fitness obtained with 360-bit genotypes

5 Discussion

5.1 Readability and Minimality of Representation

One important advantage of ATNoSFERES with respect to XCSM is that the ATN resulting from the evolution is very easy to understand. But this feature is not only a question of graphical representation.

XCSM produces a constant size list of classifiers into which the size of the external conditions part and of the memory register must be chosen in advance. As a result, there are generally more classifiers and more internal states than necessary.

By contrast, ATNoSFERES builds a graph whose number of nodes, edges, and labels on the edges are not given in advance. Thus it can build a minimal controller to solve the given problem.

Another key difference is that, in XCSM, the sequence of internal states of the agent during one run is not explicitely stated and must be derived by hand through careful examination. On the contrary, this sequence is perfectly clear when one reads an ATN. Furthermore, the internal state is very stable in ATNoSFERES. But this advantage of ATNoSFERES has its counterpart

that will be discussed in § 5.2: ATNoSFERES cannot represent `Condition-Action` rules that can be fired whatever the internal state is, as it is the case in XCSM with an internal condition composed of "#" only.

5.2 Generalization

An important difference between XCSM and ATNoSFERES formalisms call upon the elements on which generalization can take place. In the current implementation of ATNoSFERES, generalization is not possible with respect to the internal conditions and actions. This prevents ATNoSFERES from dealing with a default behavior, regardless of the internal state.

In XCSM, a `#` in the internal condition allows the classifier to be applied whatever the internal state represented by the memory register is. This mechanism permits to act regardless of the internal state.

Furthermore, in the current implementation of ATNoSFERES, there is no explicit selection pressure on the generality of the conditions on the labels, while the production of generalized classifiers is inherent to the LCS approach. Thus, we do not necessarily obtain general rules and the condition labels still contain redundant information, *e.g.* in the identification of the NE corner.

However, the conditions that are actually encountered in the graphs are quite general. In fact, once a good solution has been found, the population tends to become homogeneous and the size of genotypes stabilizes. Many different genotypes can lead to similar behaviors, but we assume that there is a bias towards compact solutions.

5.3 Reinforcement Learning and Classifier Selection

Another important difference between the ATN produced by ATNoSFERES and the list of classifiers produced by XCSM is that in the latter each classifier is endowed with a prediction representing its propensity to be fired, while in the former the edges get an equal probability to be selected if their condition token matches with the current situation.

Thus, in ATNoSFERES, if two edges can be selected simultaneously, the selection will not be deterministic. Since the optimal behavior is compatible with non-determinism only if both behaviors are strictly equivalent, the selection pressure in ATNoSFERES will prevent non-determinism in situations where it is detrimental. This provides a strong bias towards minimal controllers.

By contrast, in XCSM, several classifiers can match with the same situation, but only the strongest will be fired. Thus, it is not necessary that the other matching classifiers are deleted.

However, one important advantage of LCS with respect to ATNoSFERES is that the strength of classifiers are learned through a RL algorithm. Combining GA with RL is well known to help finding better individuals faster. In the Markov decision process (MDP) context, RL algorithms use more information about the experience of the agent than GA. While the GA only selects agents according to a global fitness function, RL algorithms distribute the reward obtained when the goal is reached only to the rules which have contributed to the behavior, taking into account the exact sequence of actions performed by the agent in the way the reward is back-propagated.

In order to remedy the fact that ATNoSFERES does not use RL, it has been necessary to include into the fitness function elements that carry some information about the actual behavior of the agent (see §4.2). But tuning such a fitness function is both difficult and crucial for the success of the experiment.

5.4 Optimality

The behaviors that have been obtained are still not completely optimal: when the agent starts from the west corridor, it should recognize the NW corner and then go directly in the third vertical corridor without checking the NE corner as it does. This is partly due to the fitness function we used: part of the time lost in exploring the NE corner is balanced by the exploration reward. Additionally, the structure for recognizing the NW corner would require at least two nodes and five edges and associated condition/action tokens. Thus it would constitute a major structural change in the graph with respect to the small selective advantage.

6 Conclusion and Future Work

From the perspective adopted in this paper, ATNoSFERES is similar to a *Pittsburgh* style LCS endowed with the ability to tackle non-Markov problems. By contrast with Michigan style LCS like XCSM, ATNoSFERES is deprived from any RL mechanism. We have shown that ATNoSFERES can produce controllers that are both very efficient in terms of the behavior they generate and very parsimonious in the way they specify that behavior. Thus we believe that ATNoSFERES is a good starting point to address more complex non-Markov problems than the benchmark experiment studied here.

The comparison with XCSM suggests two points in our agenda of research. First, it seems useful to investigate the possibility of adding a parameter equivalent to the classifier force, so as to combine RL with the GA already in use.

Second, it seems necessary to address the sub-optimality problem highlighted in §5.4. It seems that finding an optimal individual in the `Maze10` environment from the one presented in figure 5 requires a very expensive structural modification. As a result, it is unlikely that the GA will find this modification without further improvements in the representation or the mechanisms. In that respect, the ability of classifiers to deal with unspecified internal states seems a key advantage, and we should try to find a way to give that property to ATNoSFERES. Though this feature has not been implemented at this time in the model, it would only consist in copying the same edge on each existing node, by adding one special connection token to the genetic code.

References

[1] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, 1975.

[2] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution.* John Wiley & Sons, 1966.

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

[4] F. Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm.* Ph.D. thesis, ENS Lyon – Université Lyon I, 1994.

[5] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, Ann Arbor, MI, 1975.

[6] J. Kodjabachian and J.-A. Meyer. Evolution and Development of Neural Controllers for Locomotion, Gradient-Following, and Obstacle-Avoidance in Artificial Insects. *IEEE Transactions on Neural Networks*, 9:796–812, 1998.

[7] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[8] J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors. *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, CA, 1996. MIT Press.

[9] S. Landau, S. Picault, and A. Drogoul. ATNoSFERES: a Model for Evolutive Agent Behaviors. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems*, 2001.

[10] P. L. Lanzi. An Analysis of the Memory Mechanism of XCSM. In *Proceedings of the Third Genetic Programming Conference*, 1998.

[11] P. L. Lanzi and S. W. Wilson. Toward optimal classifier system performance in non-markov environments. *Evolutionary Computation*, 8(4):393–418, 2000.

[12] A. Lindenmayer. Mathematical Models for Cellular Interaction in Development, parts I and II. *Journal of theoretical biology*, 18, 1968.

[13] S. Luke and L. Spector. Evolving Graphs and Networks with Edge Encoding: Preliminary Report. In Koza et al. (8), pages 117–124.

[14] D. J. Montana. Strongly Typed Genetic Programming. In *Evolutionary Computation*, volume 3. 1995.

[15] S. Picault and S. Landau. Ethogenetics and the Evolutionary Design of Agent Behaviors. In N. Callaos, S. Esquivel, and J. Burge, editors, *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'01)*, volume III, pages 528–533, 2001.

[16] H.-P. Schwefel. *Evolution and Optimum Seeking.* John Wiley and Sons, Inc., 1995.

[17] R. S. Sutton and A. G. Barto. *Reinforcement Learning, an introduction.* MIT Press, Cambridge, MA, 1998.

[18] A. Tomlinson and L. Bull. CXCS. In P. Lanzi, W. Stolzmann, and S. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 194–208. Springer Verlag, Heidelberg, 2000.

[19] S. W. Wilson. ZCS, a Zeroth level Classifier System. *Evolutionary Computation*, 2(1):1–18, 1994.

[20] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[21] W. A. Woods. Transition Networks Grammars for Natural Language Analysis. *Communications of the Association for the Computational Machinery*, 13(10):591–606, 1970.

[22] X. Yao. Evolving Artificial Neural Networks. *Proceedings of the IEEE*, 87, 1999.

Coevolving different knowledge representations with fine-grained parallel Learning Classifier Systems

Xavier Llorà and Josep M. Garrell
Enginyeria i Arquitectura La Salle, Universitat Ramon Llull
Psg. Bonanova 8, 08022-Barcelona, Catalonia, Spain
{xevil,josepmg}@salleURL.edu

Abstract

This paper deals with the coevolution of different knowledge representations using fine-grained parallel learning classifier systems for data mining tasks. The objective is to demonstrate that a fine-grained parallel classifier systems can evolve individuals codifying different knowledge representations at the same time. This goal is achieved exploiting spatial relations of fine-grained parallel algorithms to favor the coevolution of knowledge representations, as well as extinction patterns. Experiments were performed with GALE2, a fine-grained parallel learning classifier system. Experiments focused on the diversity of the coevolved individuals, their classification accuracy, and the usefulness of the method proposed.

1 INTRODUCTION

The goal of a data mining process for classification tasks is the extraction of a certain knowledge from a given data set. The knowledge obtained from a data set ($\mathcal{P}$) to be mined is usually expressed in a certain formal language or representation. The knowledge representation used by data mining algorithms may differ between approaches. For instance, common knowledge representations for data mining are rules or decision trees, among others. On the other hand, some data mining algorithms are specially tailored for a given knowledge representation, constraining the scope of their application. Fine-grained parallel learning classifier systems can overcome this situation. Furthermore, they provide a knowledge-independent model for data mining [Llorà and Garrell, 2001c]. This paper explores how this kind of classifier systems can coevolve different knowledge representations simultaneously.

Learning classifier systems, like XCS [Wilson, 1995], have been applied to data mining problems, often looking for rule sets induction. Some examples of the application of XCS to data mining problems can be found in [Wilson, 2000, Saxon and Barry, 2000, Bernadó et al., 2001]. But, there has also been some attempts to introduce changes into the knowledge representation used by XCS, using, for instance, Lisp-like s-expressions as the condition part of the rules [Lanzi and Perrucci, 1999, Lanzi, 2001]. On the other hand, fine-grained parallel learning classifier systems, like GALE [Llorà and Garrell, 2001c], differ from this type of learning classifier systems using evolutionary models that exploits knowledge independence.

This paper explores how different knowledge representations can be coevolved in a fine-grained learning classifier scheme. This characteristic is useful when dealing with data mining problems. Presenting the knowledge using different representations may help further understanding and usage of the knowledge mined. Thus, in order to achieve this goal, GALE is modified to deal with heterogeneous runs, where individuals of the population codify different knowledge representations in their genotypes. However, the coevolution of different knowledge representations at the same time has some problems that must be taken into account. Among others, knowledge representations usually require different amounts of time to find a solution. Therefore, the knowledge representations that can achieve a solution (or a local optima) rapidly may over-take the space in the board. But, these solutions may not be the best ones in the long term run.

Therefore, this approach leads to a fine-grained parallel learning classifier system (GALE2) that exploits: (1) spatial relations to favor the coevolution of indi-

viduals using a board, and (2) extinction patterns to avoid local optima [Kirley and Green, 2000] and the take over of the board for a given knowledge representation. This paper focuses on extinction patterns and explores how they can help GALE2 to coevolve efficiently different knowledge representations at the same time. Using these two new contributions, GALE2 becomes a knowledge-independent data mining model capable of expressing the induced knowledge using several knowledge representations.

The rest of the paper is structured as follows. Section 2 describes the modification introduced in GALE to enable the coevolution of different knowledge representations that led to GALE2. The description presents the evolutionary model used, as well as the coevolutionary mechanisms introduced (selective neighborhood and extinction patterns). Next, section 3 presents the results obtained using GALE2 solving well-known data mining problems for classification tasks. Finally, section 4 presents some conclusions for the work presented.

2 GALE2 VERSUS GALE

Genetic and Artificial Life Environment (GALE) is a classifier scheme based on fine-grained parallel genetic algorithms. GALE was firstly introduced in [Llorà and Garrell, 2001c] as a data mining algorithm, being designed for solving classification tasks [Llorà and Garrell, 2001b, Llorà, 2002]. This section begins describing GALE2, focusing on its parallel evolutionary model and the main differences when compared to GALE. Then, the section pays attention to the knowledge representations evolved by GALE2 in this paper. Finally, section concludes discussing one of the main issues of GALE2: extinction patterns.

2.1 MODELING GALE2

GALE uses a 2D grid (board $\mathcal{T}$) form by $m \times n$ cells for spreading spatially the evolving population. Each cell ($\mathcal{T}_{ij}$) of the grid contains either one ($\zeta(\mathcal{T}_{ij}) = 1$) or zero individuals ($\zeta(\mathcal{T}_{ij}) = 0$); thus, for instance a 32×32 grid can contain up to 1024 individuals, each one placed on a different cell. Each individual ($\mathcal{T}_{ij}^{I}$) is a complete solution to the classification problem, in fact, each individual codifies the knowledge that describes the mined data set. GALE2 differs from GALE in the fact that the individuals are not homogeneous. Thus, an individual in GALE2 codifies one of the different knowledge representations available, as later explained (see section 2.2). Genetic operators are restricted to the immediate neighborhood ($\mathcal{T}_{ij}^{\nu}$) of the cell in the grid. The size of the neighborhood is r. Given a cell $\mathcal{T}_{ij}$ and $r = 1$, the neighborhood $\mathcal{T}_{ij}^{\nu}$ of $\mathcal{T}_{ij}$ is defined by the 8 adjacent cells to $\mathcal{T}_{ij}$ (being $\zeta(\mathcal{T}_{ij}^{\nu})$ the number of occupied cells in $\mathcal{T}_{ij}^{\nu}$). Thus, r is the number of hops that defines the neighborhood.

To introduce the coevolution of different knowledge representations, GALE2 uses a modified version of the neighborhood definition proposed in GALE. Selective neighborhood ($\mathcal{T}_{ij}^{\vartheta}$) is defined in terms of the whole neighborhood $\mathcal{T}_{ij}^{\nu}$, but the selective neighborhood is restricted to the cells in $\mathcal{T}_{ij}^{\nu}$ that contain individuals codifying the same knowledge representation of $\mathcal{T}_{ij}^{I}$ ($\zeta(\mathcal{T}_{ij}^{\vartheta}) \subseteq \zeta(\mathcal{T}_{ij}^{\nu})$). Changing the neighborhood definition implies a change in the way that genetic operators are used in GALE2 in comparison to GALE, as we introduce later. Every cell in GALE runs the same algorithm in parallel which summarizes as:

```
GALE2(T,P)
   FOR-EACH T_ij ∈ T
   DO IN PARALLEL
        t ← 0
        initialize T_ij
        evaluate T_ij^I using P
        REPEAT
              t ← t+1
              merge T_ij^I among T_ij^ϑ
              split T_ij^I among T_ij^ν
              evaluate T_ij^I using P
              survival of T_ij among T_ij^ϑ
              extinction of T_ij among T_ij^ϑ
                 if O(T) reaches a 100%
        UNTIL Ω(T_ij, t)
   DONE
RETURN T
```

During the initialization of each cell, GALE2 builds a random individual, as it is done in GALE. Not all the cells contain individuals (probability of occupation p_ζ), thus they can be full (with one individual) or empty. Each knowledge representation used has the same likelihood to be used in this process. $O(\mathcal{T})$ is the percentage of occupied cells in the board $\left(O(\mathcal{T}) = \frac{\sum_{ij}\zeta(\mathcal{T}_{ij})}{n \times m}\right)$. The individual $\mathcal{T}_{ij}^{I}$ is evaluated using the data set $\mathcal{P}$ to be mined. The fitness function used in GALE2 is the same used in GALE, $fit(I) = \left(\frac{I^c}{l}\right)^2$ [De Jong and Spears, 1991], being I^c the number of correctly classified instances and l the number of instances of the $\mathcal{P}$ data set. Next, the evolutionary cycle starts. This process runs until $\Omega(\mathcal{T}_{ij}, t)$ is satisfied. $\Omega(\mathcal{T}_{ij}, t)$ is satisfied when all the instances are correctly classified ($fit(I) = 1$), or a certain amount of iterations (k_{max}) are completed.

The *merge* in GALE2 crosses the individual in the cell with one individual randomly chosen among its selective neighborhood $\mathcal{T}_{ij}^{\vartheta}$, with a given probability $p_{\mathcal{M}}$, instead of using the whole neighborhood $\mathcal{T}_{ij}^{\nu}$ proposed by GALE. This implementation used in GALE2 leads to restricted mating among individuals codifying the same type of knowledge representation. *Merge* generates only one individual that replaces the individual in the cell, as later explained (see section 2.2).

Then, *split* is applied with a given probability $p_s(\mathcal{T}_{ij}^{I}) = k_{sp} \cdot fit(\mathcal{T}_{ij}^{I})$, being $k_{sp} \in [0,1]$ the maximum splitting rate. In GALE2, *split* works in the same way as proposed in GALE, but combining the whole and the selective neighborhood information. *Split* clones and mutates the individual in the cell. The new individual is placed in the empty cell $\mathcal{T}_{kl}$ of the whole neighborhood $\mathcal{T}_{kl} \in \mathcal{T}_{ij}^{\nu}$ with higher number of occupied cells in its whole neighborhood ($\max(\zeta(\mathcal{T}_{kl}^{\nu}))$. If all cells of the whole neighborhood are full $\left(\zeta\left(\mathcal{T}_{ij}^{\nu}\right) = 8\right)$, the new individual is placed in the cell of the selective neighborhood $\mathcal{T}_{ij}^{\vartheta}$ that contains the worst individual (lower fitness).

The last step in the evolutionary cycle, *survival*, decides if the individual is kept for the next cycle or not. This process uses the neighborhood information. If a cell has up to one neighbor ($\zeta(\mathcal{T}_{ij}^{\nu}) \leq 1$), then the probability of survival of the individual is $p_{sr}^{\zeta(\mathcal{T}_{ij}^{\nu})\leq 1}(\mathcal{T}_{ij}) = fit(\mathcal{T}_{ij}^{I})$, as proposed in GALE. Else if a cell has seven or eight neighbors $\zeta(\mathcal{T}_{ij}^{\nu}) \geq 7$ then $p_{sr}^{\zeta(\mathcal{T}_{ij}^{\nu})\geq 7}(\mathcal{T}_{ij}) = 0$, where the individual is replaced by the best selective neighbor in $\mathcal{T}_{ij}^{\vartheta}$. This method, introduced in GALE2, proposes a restricted selective pressure among individuals codifying the same knowledge representation. On the other neighborhood configurations ($1 < \zeta(\mathcal{T}_{ij}^{\nu}) < 7$), an individual survives if and only if $fit(\mathcal{T}_{ij}^{I}) \geq \overline{\mu}_{nei}^{\vartheta} + k_{sr} \times \sigma_{nei}^{\vartheta}$; $\overline{\mu}_{nei}^{\vartheta}$ is the average fitness value of the occupied selective neighbor cells $\mathcal{T}_{ij}^{\vartheta}$, and σ_{nei}^{ϑ} their standard deviation. k_{sr} is a parameter that controls the survival pressure over the current cell.

2.2 KNOWLEDGE REPRESENTATION

The evolutionary model of GALE2 coevolves different knowledge representations. In this paper, GALE2 coevolves three different knowledge representations in its heterogeneous runs: (1) sets of fully-defined instances [Llorà and Garrell, 2001b], (2) orthogonal decision trees [Quinlan, 1993], and (3) oblique decision trees [Breiman et al., 1984, Van de Merckt, 1993]. Rules can be extracted from orthogonal decision trees. Instance sets are evolved sets of instances that describe the set $\mathcal{P}$ mined, based on *nearest neighbor* algorithms. *Merge* uses two-point crossover [De Jong and Spears, 1991], and splitting is done using mutation based on generating some new values for genes randomly. The other two evolved knowledge representation are based on decision trees, codified as dynamic trees [Llorà and Garrell, 2001a]. The genetic operators used are one point crossover and random constants perturbation [Koza, 1992].

2.3 EXTINCTION PATTERNS

The last modification introduced by GALE2 is the usage of extinction patterns. The idea behind these patterns is the deletion of individuals from $\mathcal{T}$ leaving some room. The goal is to help the evolutionary algorithm to avoid local optima and the over-take of the space in $\mathcal{T}$ by a single type of knowledge representation. This idea is similar to the work proposed by [Kirley and Green, 2000], although they were solving optimization problems using *cellular genetic algorithms* [Whitley, 1993]. Nevertheless, extinction patterns have to favor the diversity across the board, and ensure the coevolution of all the knowledge representations used. This is a key point if we want to coevolve all the knowledge representations at the same time, without losing any of them along the evolutionary path.

There are several ways to approach to extinction patterns. This paper explores two different types of extinction patterns: (1) lower bound extinction patterns, and (2) upper bound extinction patterns. Lower bound extinction patterns bias board evolution toward selective neighborhoods with higher connection degrees $\left(\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right)\right)$. On the other hand, upper bound extinction patterns favors selective neighborhoods with lower connection degrees. Section 3 discusses, among others, which one of these patterns is the most suited for the coevolution of different knowledge representations.

The extinction patterns used by GALE2 are applied when the occupation of the board ($O(\mathcal{T})$) reaches 100%. We introduce two kinds of extinction patterns: (1) lower bound extinction patterns defined as $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \leq k$, and (2) upper bound extinction patterns $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \geq k$, being $k \in [0,8]$. Once the board $\mathcal{T}$ reaches full occupation, each cell $\mathcal{T}_{ij}$ test the extinction pattern used. If it is satisfied, the individual $\mathcal{T}_{ij}^{I}$ is deleted, leaving the cell empty, $\zeta(\mathcal{T}_{ij}) = 0$. For a given run, GALE2 uses only one extinction pattern. For instance, if the extinction pattern were $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \leq 4$, this test would delete all the individuals that were kept in cells that satisfy that they have less than five selective neighbors.

3 RESULTS

This section focuses on the coevolution of several knowledge representations using GALE2. The results presented in this section do not deal with the generalization capabilities of classification accuracy (in terms of cross-validation runs) of the algorithm. Some previous work in this direction using GALE can be found in [Llorà and Garrell, 2001c]. This previous work evaluate the competence of GALE when compared to well known classifiers like XCS [Wilson, 1995], C4.5 [Quinlan, 1993], or IBL [Aha and Kibler, 1991], among others. Instead, the experiments conducted in this paper look inside the evolutionary process focusing on the coevolution of individuals that encode different knowledge representations. Moreover, the experimental runs were also prepared to study the impact of the extinction patterns, presented in the previous section, in the behavior of the coevolution that takes place in the board $\mathcal{T}$ of GALE2.

In the experiments, GALE2 coevolved simultaneously the three different knowledge representations presented in the previous section. This means that an individual in a run encode in its genotype either an orthogonal decision tree, or an oblique decision tree, or a set of fully-defined instances. In order to illustrate the behavior of GALE2, it was used to solve two well-known data sets provided by the UCI repository [Merz and Murphy, 1998]: (1) the *Iris* data set (`irs`) , and (2) the *Wisconsin Breast Cancer* data set (`wbc`). A deeper analysis using other data sets is part the further work of this paper. Thus, the experiments were designed to show the usefulness of using extinction patterns in GALE2. If they are not used, GALE2 behavior is constrained by the spatial distribution of individuals at the initialization phase, being unable to guarantee the right coevolution of all the knowledge representations available.

Table 1 summarizes the results obtained when the extinction patterns presented in the previous section are used. For each extinction pattern, GALE2 was run 50 times using different random seeds, averaging the results obtained. This table presents, on the left hand side, the results obtained for `irs` data set, whereas on the right hand side, table shows the results for the `wbc` data set. Results for each data set are summarized in terms of the extinction pattern used. Lower bound extinction patterns, defined as $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \leq k$, are presented at the top, whereas the bottom of the table presents the results provided by the upper bound extinction patterns $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \geq k$. Each row in the tables shows the accuracy of the individuals in board $\mathcal{T}$, as well as the number of spatial demes and the board occupation $O\left(\mathcal{T}\right)$ for different k values. A spatial deme is defined as the set of individuals that are kept in connected cells that contain the same type of individuals. One cell c_0 is connected to another cell c_n if there is a set of cells $\{c_0, c_1, c_2 \ldots c_n\}$ that satisfies that $c_i \in c_{i+1}^{\vartheta}$ given $i = \{0, 1, \ldots n-1\}$.

The lower bound extinction patterns, $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \leq k$, present a clear behavior. When the extinction pressure increases (k gets close to 8), the diversity of demes in $\mathcal{T}$ falls. Patters where $k \geq 4$ produce less than three demes. Therefore, $\mathcal{T}$ does not contain individuals for all the knowledge representations available. These extinction patterns favor that the most rapidly suited spatial deme takes over the board. This fact holds when we take into count the mean accuracy of the population, as it can be seen in table 1. Nevertheless, these results show that the lost of diversity is a serious drawback for the coevolution of different knowledge representations.

On the other hand, upper bound extinction patterns, $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \geq k$, present a different behavior. When the extinction pressure increases (k gets close to 0), the diversity holds. This is the result of favoring demes with a small connection degrees. This fact can be observed on the amount of demes kept in $\mathcal{T}$, and in the accuracy of the board. Special mention must be done on the extinction pattern $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \geq 4$. This pattern produces the larger number of accurate spatial demes. This fact was observed in both problems, `irs` and `wbc`. Thus, the balance between diversity and uniformity, proposed by upper bound patterns, produce in GALE2 rich boards. The worth of these boards is that they are examples of how different demes can be efficiently coevolve at the same time for all the knowledge representations available, without destroying diversity. Therefore, as a result of the tests done, upper bound extinction patterns help the coevolution and diversity of different knowledge representations in GALE2. This issue is important for achieving the goal of coevolving different knowledge representation in data mining.

Some look inside GALE2 dynamics can be found in figures 1 and 2. These figures show how an extinction pattern can change the behavior of GALE2. The two figures are obtained using GALE2 solving the `irs` problem. The runs presented in the figures share the same parameters values (as shown in the appendix), as well as the random seed. This means that all the runs share the same behavior until the board collapses, presenting no empty cells. Then, when $O(\mathcal{T})$ reaches 100% (no cell remains empty in the board), GALE2 applies an extinction pattern (see section 2.3).

Lower bound extinction patterns tend to produce a

big extinction at the first application of the pattern (figure 1). But evolution adapts the spatial demes location contained in $\mathcal{T}$, reducing the scope of extinction in following pattern applications. But this adaptation is obtained by losing the diversity of the board, as early explained. This fact can be observed in figure 2. Each cell $\mathcal{T}_{ij}$ is represented using the following color code: white (empty cell), black (oblique decision three), dark gray (orthogonal decision tree), light gray (set of fully-defined instances). On the other hand, upper bound extinction patterns tend to produce a greater diversity, but eventually (as the extinctive pressure increases) they turn unstable leading to the total extinction of the population in $\mathcal{T}$, as shown in figure 1. Figure 2 also presents some snapshots of the board evolution using upper bound extinction patterns.

The main characteristic of GALE2 is that it can evolve several knowledge representations in the same heterogeneous run. As a data mining algorithm, it can provide different explanations for the data set being mined, helping the user to understand the problem being solved. We want to conclude this section of results showing some examples of the solutions coevolved using GALE2. The individuals presented are solutions to the `irs` problem. This problem is defined using 4 attributes (sepal length (**S.L.**), sepal width (**S.W.**), petal length (**P.L.**), and petal width (**P.W.**)), as well as three different classes (iris setosa (`set`), iris versicolor (`ver`), iris virginica (`vir`)). At the right hand side of each individual we also present its accuracy using the whole `irs`, by showing its confusion matrix. Each row in the matrix represents the class of the instance to classify, whereas each column is the predicted class by the individual. The first individual is an orthogonal decision tree ($*$ marks the leaf that misclassified one instance), whereas the second one is a set of fully-defined instances.

S.W. ≤ 3.023 : `set`
- **P.W.** ≤ 1.537
  - **P.L.** ≤ 4.957
    - **P.W.** ≤ 0.702 : `set`
    - **P.W.** > 0.702 : `ver`
  - **P.L.** > 4.957 : `vir`
- **P.W.** > 1.537
  - **P.L.** ≤ 4.957 : `vir`
  - **P.L.** > 4.957
    - **P.W.** ≤ 1.783 : `ver`*
    - **P.W.** > 1.783 : `vir`

	set	ver	vir
set	50	0	0
ver	0	50	*1
vir	0	0	49

S.W. > 3.023
- **P.W.** ≤ 1.014 : `set`
- **P.W.** > 1.014
  - **P.W.** ≤ 1.537 : `ver`
  - **P.W.** > 1.537
    - **P.L.** ≤ 4.957 : `ver`
    - **P.L.** > 4.957 : `vir`

S.L.	S.W.	P.L.	P.W.	Cls
4.665	3.608	3.573	0.414	`set`
5.574	2.844	3.444	1.952	`vir`
5.574	2.858	3.444	1.631	`ver`
4.882	2.281	5.600	1.836	`vir`
5.574	2.353	6.627	2.450	`vir`
5.574	2.844	6.627	1.302	`vir`

	set	ver	vir
set	50	0	0
ver	0	50	0
vir	0	0	50

4 CONCLUSIONS

This paper presented how different knowledge representations can be coevolved in a fine-grained learning classifier scheme. In order to achieve this goal, a previous learning classifier systems (GALE) was modified to deal with heterogeneous runs, where individuals of the population codify different knowledge representations in its genotype (GALE2). This approach leads to a classifier scheme that exploits: (1) spatial relations to favor the coevolution of individuals, and (2) extinction patterns to avoid local optima.

Results show that the coevolution of different knowledge representations is possible. Moreover, the results obtained also show that, when an adequate extinction pattern is used, accurate individuals belonging to different knowledge representations can be coevolved efficiently. Upper bound extinction patterns also help GALE2 to avoid that a particular type of knowledge representation over-take the space of the board. Experiments show that upper bound extinction patterns tend to favor diversity (e.g. $\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \geq 4$).

GALE2 also shows that with few more efforts, it performs as GALE. Nevertheless, GALE2 can effectively coevolve different knowledge representations at the same time reducing the number of homogeneous runs previously needed by GALE (one for each knowledge representation). Therefore, this leads to an important reduction of the resources needed (using the same parameter configuration for GALE and GALE2). On the other hand, as a data mining tool, GALE2 has the advantage, when compared to other approaches, of showing the user different kinds of solutions, favoring a deeper look at the knowledge mined.

Acknowledgments

We would like to thank the support provided by *Generalitat de Catalunya* (*DURSI* under grant number 1999FI-00719), as well as the support of *Enginyeria i Arquitectura La Salle* to our research group. We are also grateful to the Illinois Genetic Algorithms Lab people and to his director, David E. Goldberg, for the the challenging time that Xavier spent there. Finally, we also want to thank the anonymous reviewers for their useful comments that let us greatly impro-

Table 1: Results obtained using the `irs` (left) and `wbc` (right) data sets

$\zeta(\mathcal{T}_{ij}) \leq$	Accuracy	#Demes	Occupation
0	81.87±3.12	37.16±4.89	100.00±0.00
1	85.60±3.32	18.32±3.48	100.00±0.00
2	88.11±3.58	12.62±2.90	100.00±0.00
3	92.11±3.26	5.02±1.50	99.99±0.03
4	96.57±1.87	2.14±0.76	99.94±0.18
5	97.28±1.30	1.94±0.65	99.53±0.67
6	97.09±1.55	1.82±0.56	98.88±1.76
7	97.26±1.56	1.58±0.53	98.05±3.42
$\zeta(\mathcal{T}_{ij}) \geq$	Accuracy	#Demes	Occupation
8	80.98±3.20	43.30±6.08	98.93±1.23
7	81.08±3.48	40.42±5.93	99.96±0.09
6	83.55±3.47	30.46±5.41	100.00±0.00
5	83.98±3.06	37.44±6.06	99.99±0.01
4	84.19±3.40	85.78±23.24	94.61±8.51
3	87.37±6.50	14.62±8.04	54.54±36.50
2	42.34±29.87	0.94±1.28	22.54±1.50
1	1.68±8.36	0.10±0.51	0.68±0.01

$\zeta(\mathcal{T}_{ij}) \leq$	Accuracy	#Demes	Occupation
0	72.77±4.16	29.18±7.89	100.00±0.00
1	76.34±3.27	32.20±4.32	100.00±0.00
2	84.24±2.80	14.82±3.23	100.00±0.00
3	87.32±2.67	6.30±1.73	98.56±0.04
4	93.24±2.33	1.96±0.57	99.62±0.02
5	95.98±1.20	2.01±0.40	99.04±0.05
6	96.66±0.01	1.96±0.60	98.80±0.12
7	96.42±0.01	1.92±0.60	97.99±0.24
$\zeta(\mathcal{T}_{ij}) \geq$	Accuracy	#Demes	Occupation
8	90.58±0.01	40.02±5.07	99.60±0.43
7	90.33±0.01	37.82±5.00	99.99±0.03
6	91.36±0.01	28.90±5.64	100.00±0.00
5	91.03±0.01	37.64±6.02	100.00±0.00
4	87.80±0.01	124.80±10.91	99.99±0.03
3	92.65±0.01	19.54±10.81	69.47±30.36
2	56.82±0.28	1.00±0.97	23.06±19.19
1	0.00±0.00	0.00±0.00	0.00±0.00

ve the quality and the clarity of this paper.

Appendix

In order to allow the replication, the parameters of GALE2 were set as follows: m×n=32×32, k_{max}=100, p_ζ=.4, $p_{\mathcal{M}}$=.4, k_{sp}=.5, p_{mu}=.003, k_{sr}=-.25. Discussion about parameter setting can be found in [Llorà, 2002].

References

[Aha and Kibler, 1991] Aha, D. and Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

[Bernadó et al., 2001] Bernadó, E., Llorà, X., and Garrell, J. M. (2001). XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six other Learning Algorithms on Classification Tasks. *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS-2001)*, page to appear.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth International Group.

[De Jong and Spears, 1991] De Jong, K. A. and Spears, W. M. (1991). Learning Concept Classification Rules Using Genetic Algorithms. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 651–656.

[Kirley and Green, 2000] Kirley, M. G. and Green, D. G. (2000). An Empirical Investigation of Optimization in Dynamic Environments Using Cellular Genetic Algorithms. *Genetic and Evolutionary Computation Conference (GECCO2000)*, pages 11–18.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programing: On the Programing of Computers by Means of Natural Selection (Complex Adaptive Systems)*. MIT Press.

[Lanzi, 2001] Lanzi, P. L. (2001). Mining Interesting Knowledge from Data with XCS Classifier System. In *Proceedings of the Genetic and Evolutinary Computation Conference (GECCO 2001)*, pages 958–965. Morgan Kauffmann.

[Lanzi and Perrucci, 1999] Lanzi, P. L. and Perrucci, A. (1999). Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In *Proceedings of the Genetic and Evolutinary Computation Conference (GECCO'99)*, pages 345–352. Morgan Kauffmann.

[Llorà, 2002] Llorà, X. (February, 2002). *Genetic Based Machine Learning using Fine-grained Parallelism for Data Mining*. PhD thesis, Enginyeria i Arquitectura La Salle. Ramon Llull University, Barcelona, Catalonia, European Union.

[Llorà and Garrell, 2001a] Llorà, X. and Garrell, J. M. (2001a). Evolution of Decision Trees. In *4th Catalan Conference on Artificial Intelligence (CCIA'2001)*, pages 115–122. Morgan Kauffmann.

[Llorà and Garrell, 2001b] Llorà, X. and Garrell, J. M. (2001b). Evolving Partially-Defined instances with Evolutionary Algorithms. In *Proceedings of the 18th International Conference on Machine Learning (ICML'2001)*, pages 337–344. Morgan Kauffmann.

[Llorà and Garrell, 2001c] Llorà, X. and Garrell, J. M. (2001c). Knowledge-Independent Data Mining with Fine-Grain Parallel Evolutionary Algorithms. In *Proceedings of the Genetic and Evolutinary Computation Conference (GECCO 2001)*, pages 461–468. Morgan Kauffmann.

[Merz and Murphy, 1998] Merz, C. J. and Murphy, P. M. (1998). UCI Repository for Machine Learning Data-Bases [http://www.ics.uci.edu/~mlearn/MLRepository. html]. *Irvine, CA: University of California, Department of Information and Computer Science*.

[Quinlan, 1993] Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

[Saxon and Barry, 2000] Saxon, S. and Barry, A. (2000). XCS and the Monk's Problems. In Lanzi, S. and Wilson, editors, *Learning Classifier Systems: From Foundations to Applications*, pages 223–242.

[Van de Merckt, 1993] Van de Merckt, T. (1993). Decision trees in numerical attribute spaces. In *Proceedings of the 13th Intenational Joint Conference on Artificial Intelligece*, pages 1016–1021. Morgan Kaufmann.

[Whitley, 1993] Whitley, D. (1993). Cellular Genetic Algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms*, page 658.

[Wilson, 1995] Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.

[Wilson, 2000] Wilson, S. W. (July, 2000). Mining Oblique Data with XCS. *IlliGAL Report No. 2000028*.

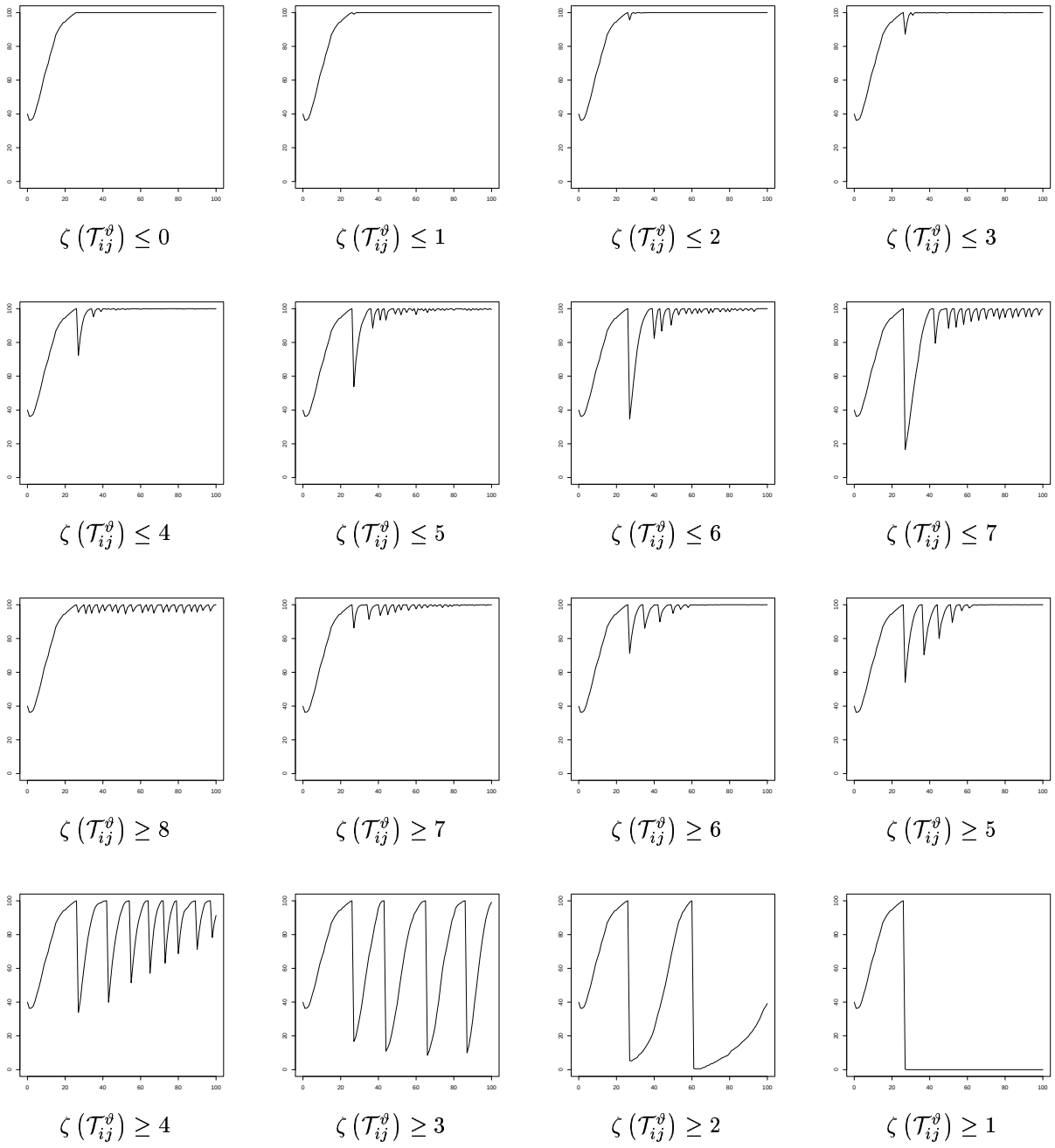


Figure 1: Board occupation for the `irs` problem using different extinction patterns. Each figure presents the board occupation percentage behavior, $O(\mathcal{T})$, through the run. The only difference between runs is the extinction pattern used. Therefore, all the runs share the same occupation behavior until iteration `t=26`. After the full board occupation, the extinction pattern is applied, leading to different evolutionary paths. The pattern used is shown below each figure. These curves are the ones obtained in the runs also presented in figure 2. As it can be seen, lower bound extinction patterns produce steady occupation of the board. On the other hand, upper bound extinction patterns produce oscillating occupation of the board favoring diversity, but eventually leading to a total extinction when extreme extinction pressure is applied.

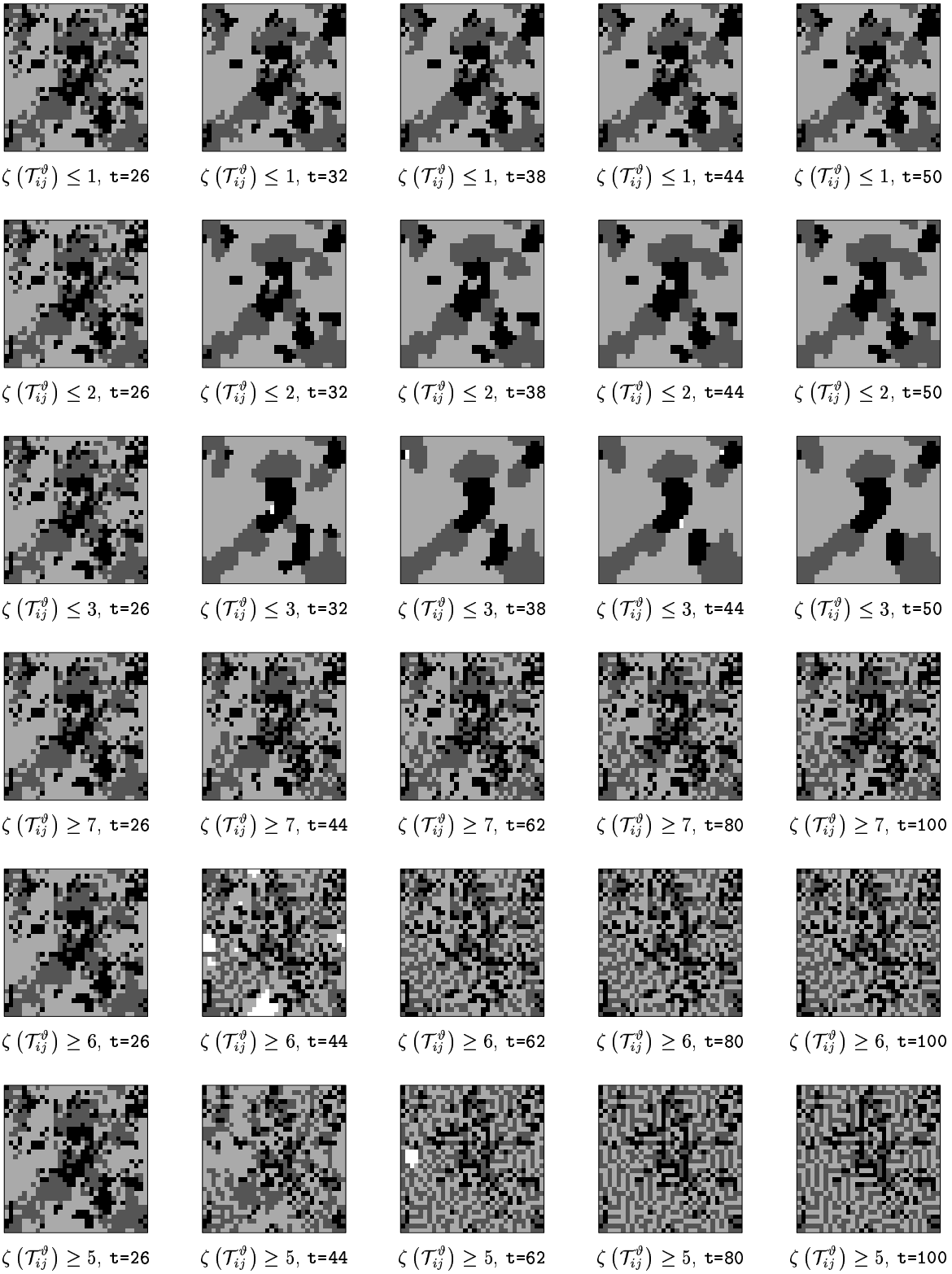


Figure 2: Board evolution for the `irs` problem using lower $\left(\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \leq k\right)$ and upper $\left(\zeta\left(\mathcal{T}_{ij}^{\vartheta}\right) \geq k\right)$ bound extinction. Color code: white (empty cell), black (oblique decision three), dark gray (orthogonal decision tree), light gray (set of fully-defined instances).

Hyper-heuristics: learning to combine simple heuristics in bin-packing problems

Peter Ross
School of Computing
Napier University
Edinburgh EH10 5DT
peter@dcs.napier.ac.uk

Sonia Schulenburg
School of Computing
Napier University
Edinburgh EH10 5DT
s.schulenburg@napier.ac.uk

Javier G. Marín-Blázquez
Division of Informatics,
The University of Edinburgh
Edinburgh EH1 1HN, UK
javierg@dai.ed.ac.uk

Emma Hart
School of Computing
Napier University
Edinburgh EH10 5DT
emmah@dcs.napier.ac.uk

Abstract

Evolutionary algorithms (EAs) often appear to be a 'black box', neither offering worst-case bounds nor any guarantee of optimality when used to solve individual problems. They can also take much longer than non-evolutionary methods. We try to address these concerns by using an EA, in particular the learning classi£er system XCS, to learn a solution process rather than to solve individual problems. The process chooses one of various simple non-evolutionary heuristics to apply to each state of a problem, gradually transforming the problem from its initial state to a solved state. We test this on a large set of one-dimensional bin packing problems. For some of the problems, none of the heuristics used can £nd an optimal answer; however, the evolved solution process can £nd an optimal solution in over 78% of cases.

1 INTRODUCTION

Heuristic algorithms are very widely used to tackle practical problems in operations research, because so many are NP-hard [12] and exhaustive search is often computationally intractable. Evolutionary algorithms (EAs) can be excellent for searching very large spaces, at least when there is some reason to suppose that there are 'building blocks' to be found. A 'building block' is a fragment, in the chosen representation, such that chromosomes which contain it tend to have higher £tness than those which don't. EAs bring building blocks together by chance recombination, and building blocks which are not present in the population at all may still be generated by mutation.

However, the use of EAs are often justi£ed simply by results. If you knew what building blocks looked like in advance, you would not need an EA to bring them together. Nor, usually, are there performance guarantees: in epistatic problems it can happen that the best solutions cannot simply be fabricated from good-looking building blocks. Because of this, EAs often have a problem of acceptability – they look like a 'black box' algorithm that you run until it delivers a solution, but you often do not know whether that solution is even close to optimal, and while it is running you have no easy way to forecast properties of the outcome. The delivered solution may also be fragile, in the sense that there is little continuity between problem speci£cation and EA solution: if you change the problem only slightly, the solution found by re-running the EA changes drastically. It is not surprising therefore that in many practical applications, people may prefer to use a simple heuristic that is comprehensible and perhaps also offers worst-case performance guarantees.

This paper represents a step towards a new way of using EAs that may solve some problems of acceptability for real-world use. The basic idea is as follows: instead of using an EA to discover a solution to a speci£c problem, we use an EA to try to fabricate a solution process applicable to many problem instances and built from simple, well-understood heuristics. Such a solution process might consist of using a certain heuristic initially, but after a while the nature of the remainder of the task may be such that a different heuristic becomes more appropriate.

For example, in [21] an early version of this idea was used to tackle large exam timetabling problems by choosing two heuristics and associated parameters, together with a test for when to switch from using the £rst to using the second. This was motivated by the unsurprising observation that different academic institutions have very different constraints. One institution might have some very large exams, limited exam seating and many smaller exams, so that the important task early on is to pack those large exams together as far as possible in order to plenty of space to deal with placing the many smaller exams. Another institution might have no very large exams, but instead the exams can be clustered such that there are very few inter-cluster con-

straints, and exam clusters can therefore be viewed as relatively independent sub-problems, for which you might naturally choose some other heuristic that placed little emphasis on packing large exams.

An obvious objection to the general idea of hyper-heuristic methods is that, if you combine the use of several heuristics when solving a problem, you will probably lose any worst-case performance guarantee that an individual heuristic had. But against this, there is a simple way to judge the ef£cacy of a composite algorithm against the use of any single heuristic – you might be able to seed the initial EA population with a few chromosomes that represented the process of using only a single heuristic from start to £nish. If such chromosomes do not survive, it is because composite algorithms outperformed them.

In what follows, we describe an example of using hyper-heuristic methods to tackle one-dimensional bin-packing problems. A modern learning classi£er system, XCS [22], is used to learn a set of rules which associate characteristics of the current state of a problem with speci£c heuristics. The set of rules is used to solve problems as follows: given the initial problem characteristics P, a heuristic H is chosen and it packs a bin, thus gradually altering the characteristics of the problem that remains to be solved. At each step a rule appropriate to the current problem state P' is chosen, and the process repeats until all items have been packed.

Using any Michigan-style classi£er system means, of course, that we cannot do what we suggested above and inject pure heuristics into the initial population in order to compare them against composite ones. In a Michigan-style system, the whole population represents one composite algorithm. Nevertheless, XCS represents a simple way to try to fabricate a composite algorithm and the interest lies in seeing how well it can work. In particular, if the system is trained using a few problems, does it then generalise by also performing well on lots of unseen problems? If so (and, to spoil the story, the answer given below is 'yes'), then this is a useful step towards the concept of using EAs to generate strong solution processes rather than merely using them to £nd good individual solutions.

The approach is tested using a large set of benchmark one-dimensional bin-packing problems and a small set of eight heuristics. No single one of the heuristics used is capable of £nding the optimal solution of more than a very few of the problems; however, the evolved rule-set was able to produce an optimal solution for over 78% of them, and in the rest it produced a solution very close to optimal.

2 ONE-D BIN-PACKING

In the one-dimensional Bin Packing problem (BPP1), there is an unlimited supply of bins, each with capacity c (a positive number). A set of n items is to be packed into the bins, the size of item i is $s_i > 0$, and items must not over-£ll any bin:

$$\sum_{i \in \text{bin}(k)} s_i \leq c$$

The task is to minimise the total number of bins used. Despite its simplicity, this is an NP-hard problem. If M is the minimal number of bins needed, then clearly:

$$M \geq \lceil (\sum_{i=1}^{n} s_i)/c \rceil$$

and for any algorithm that does not start new bins unnecessarily, $M \leq$ bins used $< 2M$ (because if it used $2M$ or more bins there would be two bins whose combined contents were no more than c, and they could be combined into one).

Many results are known about speci£c algorithms. For example, a commonly-used algorithm is First-Fit-Decreasing (FFD): items are taken in order of size, largest £rst, and put in the £rst bin where they will £t (a new bin is opened if necessary, and effectively all bins stay open). It is known [15] that this uses no more than $11M/9+4$ bins. A good survey of such results can be found in [6]. A good introduction to bin-packing algorithms can be found in [18], which also introduced a widely-used heuristic algorithm, the Martello-Toth Reduction Procedure (MRTP). This simply tries to repeatedly reduce the problem to a simpler one, by £nding a combination of 1-3 items that provably does better than anything else (not just any combination of 1-3 items) at £lling a bin, and if so packing them. This may eventually halt with some items still unpacked; the remainder are packed using a 'largest £rst, best £t' algorithm.

Various authors have applied EAs to bin-packing, notably Falkenauer's grouping GA [9, 11, 10]; see also [16, 19] for different approaches. For example, Reeves [19] used a GA to £nd the order in which to feed items to a sequential heuristic such as First-Fit, with reasonable success on a subset of the problems we use in this paper. Falkenauer also produced two classes of benchmark problems. In one of these, the so called *triplet problems*, every bin contains three items; they were generated by £rst constructing a solution which £lled every bin exactly, and then randomly shrinking items a little so that the total shrinkage was less than the bin capacity (thus the same number of bins is necessary).

As ever, speci£c knowledge about problems can help greatly. Suppose you know in advance that each bin contains exactly three items. Take items in order, largest £rst, and for each item search for two others that come very close to £lling the bin. A backtracking algorithm that considers such '£ller pairs', taking pairs in which the two members at most nearly equal in size £rst and permitting only

limited backtracking, solves many of the Falkenauer triplet problems very quickly. See [13] for some questions about whether these problems are hard or not.

The reader may wonder if the simple strategy of searching for a combination of items which come as close as possible to £lling a bin, thereby reducing the problem to a simpler one in which there seems to be more available slack, is a good one. But consider a problem in which bins have capacity 20 and there are six items: 12, 11, 11, 7, 7, 6. One bin can be completely £lled ($7+7+6$) but then three more bins are needed since the three largest items are each larger than half a bin. If bins are under-£lled, then a three-bin solution is possible, for example $12+7$, $11+7$, $11+6$. We hope this will help to convince the reader that even one-dimensional bin-packing problems have their interest. And they are worth studying because bin-packing is a constituent task of many other optimisation problems; exam timetabling is just one such example.

3 ABOUT XCS

Learning classi£er systems of the Michigan type evolve a set of condition-action rules, by measuring the performance of individual rules and then periodically using crossover and mutation to breed new rules from old. An early account can be found in [14], a more modern account and recent work is in [17].

In early learning classi£er systems, rules occasionally did an action that earned external reward, and this contributed to the rule's £tness and to the £tness of those that enabled it to £re. Earned rewards were spread by the so-called 'bucket brigade algorithm' (effectively a trickle-down economy) or 'pro£t-sharing plan' (essentially a communal reward-sharing) or other such algorithm. However, in those early systems, a rule's £tness was a measure of the reward it might earn (when considering what rule to £re) and also a measure of the reward it had earned (when selecting rules for breeding). This caused various problems, notably that rules which £red very rarely but were crucial when they did would tend to be squeezed out of the population by the evolutionary competition long before they could demonstrate their true value. XCS [22] largely £xed this by instead valuing a rule for the accuracy rather than the size of its prediction of reward.

For this reason – because, in our application, there might be heuristics which were rarely used but crucial – we chose to use XCS rather than, say, Goldberg's SCS.

4 BIN-PACKING BENCHMARK PROBLEMS

We used problems from two sources. The £rst collection is available from Beasley's OR-Library [1], which contains problems of two kinds that were generated and largely studied by Falkenauer [10]. The £rst kind, 80 problems named `uN_M`, involve bins of capacity 150. N items are generated with sizes chosen randomly from the interval 20-100. For N in the set $(120, 250, 500, 1000)$ there are twenty problems, thus M ranges from 00 to 19. The second kind, 80 problems named `tN_M`, are the triplet problems mentioned earlier. The bins have capacity 1000. The number of items N is one of 60, 120, 249, 501 (all divisible by three), and as before there are twenty problems per value of N. Item sizes range from 250 to 499 but are not random; the problem generation process was described earlier.

The second class of problems we study in this paper comes from the Operational Research Library [2] at the *Technische Universität Darmstadt*. We used their 'bpp1-1' set and their very hard 'bpp1-3' set in this paper. In the bpp1-1 set problems are named `NxCyWz_a` where `x` is 1 (50 items), 2 (100 items), 3 (200 items) or 4 (500 items); `y` is 1 (capacity 100), 2 (capacity 120) or 3 (capacity 150); `z` is 1 (sizes in 1...100), 2 (sizes in 20...100) or 4 (sizes in 30...100); and `a` is a letter in A...T indexing the twenty problems per parameter set. (Martello and Toth [18] also used a set with sizes drawn from 50...100, but these are far too easy.) Of these 720 problems, the optimal solution is known in 704 cases and in the other sixteen, the optimal solution is known to lie in some interval of size 2 or 3. In the hard bpp1-3 set there are just ten problems, each with 200 items and bin capacity 100,000; item sizes are drawn from the range $20,000 \ldots 35,000$. The optimal solution is known in only three cases, in the other seven the optimal solution lies in an interval of size 2 or 3. These results were obtained with an exact procedure called BISON [20] that employs a combination of tabu search and modi£ed branch-and-bound.

In all, therefore, we use 890 benchmark problems.

5 COMBINING HEURISTICS WITH XCS

The £rst subsection describes the heuristics we decided to use, and why. The next subsection describes the representation used within XCS. Then we describe how XCS is used to discover a good set of rules.

5.1 The set of heuristics

We £rst evaluated a variety of heuristics to see how they performed on our benchmark collection. Of the fourteen that we tried, some were taken directly form the literature,

others were variants created by us. Some of these algorithms were always dominated by others; among those that sometimes obtained the best of the fourteen results on a problem, some were always £rst equal rather than being uniquely the best of the set. We do not have space here to describe the full set, but we chose to use four whose performance seemed collectively to be representative of the best. These were:

- FFD, described in Section 2 above. This was the best of the fourteen heuristics in over 81% of the bpp1-1 problems, but was never the winner in the bpp1-3 problems.
- Next-Fit-Decreasing (NFD): an item is placed in the current bin if possible, or else a new bin is opened and becomes the current bin and the item is put in there. This is usually very poor.
- Djang and Finch's algorithm (DJD), see [7]. This puts items into a bin, taking items largest-£rst, until that bin is at least one third full. It then tries to £nd one, or two, or three items that completely £ll the bin. If there is no such combination it tries again, but looking instead for a combination that £lls the bin to within 1 of its capacity. If that fails, it tires to £nd such a combination that £lls the bin to within 2 of its capacity; and so on. This of course gets excellent results on, for example, Falkenauer's problems; it was the best performer on just over 79% of those problems but was never the winner on the hard bpp1-3 problems.
- DJT (Djang and Finch, more tuples): a modi£ed form of DJD considering combinations of up to £ve items rather than three items. In the Falkenauer problems, DJT performs exactly like DJD, as we would expect; in the bpp1-1 problems it is a little better than DJD.

In addition we also used these algorithms each coupled with a '£ller' process that tried to £nd any item at all to pack in any open bins rather than moving on to a new bin. This might, for example, make a difference in DJD if a bin could be better £lled by using more than three items once the bin was one-third full. Thus, in all we used eight heuristics. The action of the £ller process is described later.

5.2 Representing problem state for XCS

As explained above, the idea is to £nd a good set of rules each of which associates a heuristic with some description of the current state of the problem. To execute the rules, the initial state is used to select a heuristic and that heuristic is used to pack a bin. The rules are then consulted again to £nd a heuristic appropriate to the altered problem state, and the process repeats until all items have been packed.

The problem state is reduced to the following simple description. The number of items remaining to be packed are examined, and the percentage R of items in each of four ranges is calculated. These ranges are shown in Table 1. These are, in a sense, natural choices since at most one

Table 1: Item size ranges

Huge:	items over $1/2$ of bin capacity
Large:	items from $1/3$ up to $1/2$ of bin capacity
Medium:	items from $1/4$ up to $1/3$ of bin capacity
Small:	items up to $1/4$ of bin capacity

huge item will £t in a bin, at most two large items will £t a bin, and so on. The percentage of items that lie within any one of these ranges is encoded using two bits as shown in Table 2. Thus, there are two bits for each of the four

Table 2: Representing the proportion of items in a given range

Bits	Proportion of items
0 0	0 – 10%
0 1	10 – 20%
1 0	20 – 50%
1 1	50 –100%

ranges. Finally, it seemed important to also represent how far the process had got in packing items. For example, if there are very few items left to pack, there will probably be no huge items left. Thus, three bits are used to encode the percentage of the original number of items that still remain to be packed; Table 3 gives the details.

Table 3: Percentage of Items Left

Bits	% left to pack
0 0 0	0 – 12.5
0 0 1	12.5 – 25
0 1 0	25 – 37.5
0 1 1	37.5 – 50
1 0 0	50 – 62.5
1 0 1	62.5 – 75
1 1 0	75 – 87.5
1 1 1	87.5 – 100

The action is an integer indicating the decision of which strategy to use at the current environmental condition, as shown in Table 4. As mentioned earlier, the second four actions use a £ller process too, which tries to £ll any open bins as much as possible. If the £lling action successfully inserts at least one item, the £lling step £nishes. If no insertion was possible, then the associated heuristic (for example, FFD in 'Filler+FFD') is used. This guarantees a

change in the problem state. It is important to remember that the trained XCS chooses deterministically, so that it is important for the problem state (if not the state description) to change each time, to prevent endless looping.

Table 4: The action representation

Action	Meaning, Use
000	FFD
001	NFD
010	DJD
011	DJT
100	Filler + FFD
101	Filler + NFD
110	Filler + DJD
111	Filler + DJT

The alert reader might wonder whether the above problem state description in some way made heuristic selection an easy task. However, when we evaluated each of our 14 original heuristics we found many cases where two problems had the same initial state description but different algorithms were the winners of the 14-way contest. For each of the 14 algorithms we tried using a perceptron to see whether it was possible to classify problems into those on which a given algorithm was a winner and those on which it was not a winner. In every case, it was not possible, and therefore the learning task faced by XCS was not a trivial one.

6 THE EXPERIMENTS

We used Martin Butz' version of XCS [3, 4, 5] available free over the web from the IlliGAL site.

We used a single step environment, in which a reward is available at every step, and we de£ned a step as packing one bin (FFD was modi£ed to pack no more than one bin before returning). The reward earned is proportional to how well £lled that packed bin is. For example if the bin is packed to 94% of capacity, then the reward earned is 0.94. (Following the suggestion of Falkenauer and Delchambre [8], an alternative worth trying in future would be to use the square of this instead). Remember that 'packing' here means continuing to the point where the heuristic would switch bins, rather than optimally packing. Full reward is paid for packing the £nal bin. Otherwise, an algorithm which, say, placed the £nal item of size 1 in a £nal bin in order to complete the packing would earn only 0.01. The £ller is rewarded slightly differently; it is rewarded in proportion to how much it reduces the empty space in the open bins.

The XCS parameters used were exactly as used in [22], with a 50/50 explore/exploit ratio.

For training, we divided each set of bin-packing problems into a training and a test set. In each case the training set contained 75% of the problems; every fourth problem was placed in the test set. Since the problems come in groups of twenty for each set of parameters, the different sorts of problem were well represented in both training and test sets. We also combined all problems into one large set of 890 problems and divided that into a training and a test set in the same way. In the results below, we only report on what happened with this combined collection, in which the training set has 667 problems and the test set has 223 problems. Other results are omitted for space reasons; the combined set provides a good test of whether the system can learn from a very varied collection of problems.

The experiments proceeded as follows. We set a limit of L explore/exploit cycles for XCS, where the values we tried were $L = 100, 500, 1000, 5000, 10000, 25000$. During the learning phase, XCS £rst randomly chooses a problem to work on from the training set. One step (whether explore or exploit) corresponds to £lling one bin. In an explore step the action is chosen randomly, in an exploit step it is chosen according to the maximum prediction appropriate to the current problem state description. This is repeated until all the items in the current problem have been packed. A new random problem is then chosen. Clearly, a large problem such as one of the `u1000_M` will consume a great many cycles. We recorded the best result obtained on each problem during this training phase. Remember, however, that training continues, so the rule set may change after such a best result was found. In particular, the £nal rule set at the end of all training might not be able to reproduce the best result on every problem. Nevertheless, it is reasonable to record the best result found during (rather than at the end of) training on each problem, because these are still reproducible results, by re-running the training with the same seed, and easily so.

At the end of training, the £nal rule set is used on every problem in the training set to assess how well this rule set works. It is also applied to every problem in the test set.

7 RESULTS

For the problems we used, details of optimal results are available from [2] and from [1], see Section 4. In the sixteen problems where only a range is known within which the optimal number must lie, we use the upper bound.

The results were as follows:

- during training, XCS found the optimal result for 78.1% of all problems, and for all the others the best result was only one or two bins worse than optimal.

This is encouraging, because for some heuristic algorithms the performance on certain problems can be considerably worse than optimal.

- after £nding a £nal rule set, this was tested. On the training set it found the optimal result on 77.7% of problems. On the test problems, not used during training, it found the optimal result for 74.6% of problems (166 of 223) and again, results were close to optimal on all the rest.

Are these results good? The classi£er system was able to achieve the optimal result in 78.1% of all the benchmark problems, whereas the best single performer of the heuristics considered (namely, our own DJT, introduced in this paper for the £rst time) achieved only 73%. Even though these two results might seem close, it is worth noting that DJT solved none of the very hard bpp1-3 problems while the XCS-generated rule set solved seven out of the ten. It is also noteworthy that, when XCS was trained only on a training set composed of seven of the ten hard bpp1-3 problems, it solved six of those seven, and also one of the three unseen problems. In both cases no other heuristic used alone was able to solve any of these problems.

The worst heuristic is NFD; alone, it was never a winner among the original 14 heuristics we considered. We did include it in the set of heuristics that the classi£er system could invoke, and interestingly it was indeed sometimes invoked as part of a sequence that led to an optimal result, although this happened rarely.

8 CONCLUSIONS AND FUTURE WORK

This paper represents a step towards developing the concept of hyper-heuristics: using EAs to £nd powerful combinations of more familiar heuristics.

From the experiments shown it is also interesting to note that:

- XCS was able to create and develop feasible hyper-heuristics that performed well on a large collection of benchmark data sets found in literature, and better than any individual heuristic.
- The system always performed better than the worst of the algorithms involved, and in fact produced results that were either optimal (in the large majority of cases) or else were close to optimal.
- The system is able to generalise well. Results of the exploit steps during training are very close to results using a trained classi£er on new test cases. This means that particular details learned (a structure of some kind) during the adaptive phase (when the classi£er rules are being modi£ed according to experience, etc.) can be reproduced with completely new data (unseen problems taken from the test sets). For example, for one of Falkenauer's problems DJD (and our DJT) produced a new best, and optimal, result (this had already been reported in [7] where DJD was described). Even if this problem is excluded from the training set, the learned rule set can still solve it optimally.

In the work reported here we used a single-step environment (reward available after each step). It might be thought that a multi-step environment, with reward proportional to solution quality paid only at the end of a problem or at least after a number of steps were performed. However, learning is likely to be much slower, and we do not even know the number of steps needed to reach a solution in advance. In some problems, such as the `u1000_M`, we have 1000 items to pack and the number of steps to reach a solution and earn any reward could be very large.

We recognise that the reward mechanism perhaps overvalues the £lling of bins, and intend to investigate alternative reward schemes.

Other possible ways to use the multi-step environment could be to allow a chosen rule to continue to perform its action until one of the following happens:

- the problem state has changed so that the rule which chose the action is no longer applicable; or,
- a certain sizeable percentage of items have been placed, eg 20%. This would limit the chain of actions to be at most 5 steps long.

Perhaps also including some extra information about the status of the open bins might be useful. For example, if many open bins contained very little free space and there were many small items still to pack, it might be useful to be able to invoke a heuristic which tried to £ll and £nally close those bins.

Although we have focussed on bin-packing problems in this paper, similar hyper-heuristic ideas could be applied to many other kinds of problem, in which heuristics can be used step by step to transform the problem state from an initial to a £nal one. This raises interesting research questions about how sensitive the approach might be to the choice of heuristics and to the problem state description used.

Acknowledgments

This work has been supported by UK EPSRC research grant number GR/N36660.

References

[1] http://www.ms.ic.ac.uk/info.html.

[2] http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/binpp/.

[3] Martin V. Butz. An Implementation of the XCS classi£er system in C. Technical Report 99021, The Illinois Genetic Algorithms Laboratory, 1999.

[4] Martin V. Butz. XCSJava 1.0: An Implementation of the XCS classi£er system in Java. Technical Report 2000027, Illinois Genetic Algorithms Laboratory, 2000.

[5] Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. Technical Report 2000017, Illinois Genetic Algorithms Laboratory, 2000.

[6] E.G Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing, Boston, 1996.

[7] Philipp A. Djang and Paul R. Finch. Solving One Dimensional Bin Packing Problems. *Journal of Heuristics*, 1998.

[8] E. Falkenauer and A. Delchambre. A genetic algorithm for bin packing and line balancing. In *Proc. of the IEEE 1992 International Conference on Robotics and Automation*, pages 1186–1192, 1992.

[9] Emanuel Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, 2(2):123–144, 1994.

[10] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996. http://citeseer.nj.nec.com/falkenauer96hybrid.html.

[11] Emanuele Falkenauer. A Hybrid Grouping Genetic Algorithm for Bin Packing. Working Paper IDSIA-06-99, CRIF Industrial Management and Automation, CP 106 - P4, 50 av. F.D.Roosevelt, B-1050 Brussels, Belgium, 1994.

[12] Michael R. Garey and David S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. Freeman, 1979.

[13] I. P. Gent. Heuristic Solution of Open Bin Packing Problems. *Journal of Heuristics*, 3(4):299–304, 1998.

[14] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA., 1989.

[15] D.S. Johnson. *Near-optimal bin-packing algorithms*. PhD thesis, MIT Department of Mathematics, Cambridge, Mass., 1973.

[16] Sami Khuri, Martin Schutz, and Jörg Heitkötter. Evolutionary heuristics for the bin packing problem. In D. W. Pearson, N. C. Steele, , and R. F. Albrecht, editors, *Arti£cial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Ales, France, 1995*, 1995.

[17] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classi£er Systems: From Foundations to Applications*, volume 1813 of *Lecture Notes in Arti£cial Intelligence*. Springer-Verlag, Berlin, 2000.

[18] Silvano Martello and Paolo Toth. *Knapsack Problems. Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[19] C. Reeves. Hybrid genetic algorithms for bin-acking and related problems. *Annals of Operations Research*, (63):371–396, 1996.

[20] Armin Scholl and Robert Klein. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 1997.

[21] Hugo Terashima-Marín, Peter Ross, and Manuel Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 635–642. Morgan Kaufmann, 1999. early hyper-heuristic.

[22] Stewart W. Wilson. Classi£er Systems Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.