

Self-Adaptive Constructivism in Neural XCS and XCSF

Gerard David Howard
University of the West of England
Frenchay Campus
Bristol, Avon, UK
(00) 44 117 965 6261
gerard2.howard@uwe.ac.uk

Larry Bull
University of the West of England
Frenchay Campus
Bristol, Avon, UK
(00) 44 117 965 6261
larry.bull@uwe.ac.uk

Pier-Luca Lanzi
Politecnico di Milano
Piazza Leonardo da Vinci, 32
I-20133 Milano, Italy
(00) 39 02 2399 3472
pierluca.lanzi@polimi.it

ABSTRACT

For artificial entities to achieve high degrees of autonomy they will need to display appropriate adaptability. In this sense adaptability includes representational flexibility guided by the environment at any given time. This paper presents the use of constructivism-inspired mechanisms within a neural learning classifier system which exploits parameter self-adaptation as an approach to realize such behaviour. The system uses a rule structure in which each is represented by an artificial neural network. It is shown that appropriate internal rule complexity emerges during learning at a rate controlled by the system. Further, the use of computed predictions is shown possible.

Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning – *knowledge acquisition, parameter learning, connectionism and neural nets.*

General Terms

Experimentation.

Keywords

Constructivism, Learning Classifier Systems, Neural Networks, Reinforcement Learning, Self-Adaptation.

1. INTRODUCTION

The neural constructivist (NC) [14] explanation for the emergence of reasoning within brains postulates that the dynamic interaction between neural growth mechanisms and the environment drives the learning process. This is in contrast to related evolutionary selectionist ideas which emphasize regressive mechanisms whereby initial neural over-connectivity is pruned based on a measure of utility [7]. The scenario for constructivist learning is that, rather than start with a large neural network development begins with a small network. Learning then adds appropriate structure, particularly through growing/pruning dendritic connectivity, until some satisfactory level of utility is reached. Suitable specialized neural structures are not specified *a priori*; the representation of the problem space is flexible and tailored by the learner's interaction with it. We are interested in the feasibility

of a constructive approach to realize flexible learning within Learning Classifier Systems (LCS) [10], exploiting its genetic algorithm (GA) [9] foundation. In this paper we present a form of neural LCS [2] based on XCS [19] and XCSF [20]. In particular, we explore the success of extensions to the XCS-based neural LCS, N-XCS [3], including the use of self-adaptive search operators, neural constructivism (to grow hidden layer neurons), and prediction computation on versions of two well-known maze tasks.

We shall refer to the three systems presented using the following nomenclature: non-adaptive N-XCS (naN-XCS), self-adaptive N-XCS (saN-XCS), neural constructive and self-adaptive XCS (ncN-XCS). To our knowledge, this is the first implementation of XCS which uses self-adaptive parameters alongside NC to perform goal finding in simulated maze environments, as well as the first implementation of XCSF functionality within this self-adaptive, constructivist framework.

The paper is ordered as follows: the next section provides a brief overview of related work. Section 3 describes the modifications made to the XCS framework for the neural rule representation. Section 4 presents the results of naN-XCS in solving two maze environments, and compares the results with the same experiments attempted by saN-XCS, and ncN-XCS. Versions of these systems are then extended to include prediction computation, i.e., XCSF implementations are then explored.

2. RELATED WORK

The use of constructivism within neural learning classifier systems was first described by Bull [2], using Wilson's ZCS [18] as a basis. Hurst and Bull [11] later extended this work to include parameter self-adaptation and used it for real mobile robot control. In both cases it is reported that networks of different structure evolve to handle different areas of the problem space thereby identifying the underlying structure of the task. In this paper we take the principles of self-adaptation and constructivism and explore them within the accuracy-based XCS and XCSF systems.

As we are experimenting with rules based on neural networks, work on alternate representations which compute actions based on inputs are closely related: fuzzy logic (e.g., see [6] for an overview); Lisp S-Expressions [1]; and parameterized functions [13][21].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

Copyright 2008 ACM 978-1-60558-130-9/08/07...\$5.00.

3. IMPLEMENTATION

3.1 Maze environments

The mazes in traditional LCS research are encoded as binary strings that represent the local topology of the maze. The length of the string depends upon the number of exclusive object types represented in the maze. For example, a maze with three exclusive object types requires each object to be represented by two bits (e.g. 00 = empty, 01 = obstacle, 11=food) giving a 16-bit string representing the eight cells surrounding the agent. The maze environments used in this paper are the benchmarks Woods1 and Maze4 [12]. Performance is chiefly gauged by a “Step-to-goal” count – the number of discrete movements required to reach the goal state from a random starting position in the maze. In Woods1 the optimal figure is 1.69 steps and in Maze 4 it is 3.5, these are shown on the step-to-goal graphs as dashed lines. Figure 1 and 2 shows the layout of the toroidal Woods1 and bounded Maze4 respectively. Here, “O” represents an obstacle, “*” an empty space and “G” the goal state.

*	*	*	*	*
*	O	O	G	*
*	O	O	O	*
*	O	O	O	*
*	*	*	*	*

Figure 1. The Woods1 Environment

O	O	O	O	O	O	O	O
O	*	*	O	*	*	G	O
O	O	*	*	O	*	*	O
O	O	*	O	*	*	O	O
O	*	*	*	*	*	*	O
O	O	*	O	*	*	*	O
O	*	*	*	*	O	*	O
O	O	O	O	O	O	O	O

Figure 2. The Maze4 Environment

3.2 A Neural XCS

Following Bull & O’Hara [3], a number of changes were made to the standard XCS algorithm to accommodate the use of artificial neural network rules. The reader is referred to Butz & Wilson [5] for an algorithmic description of XCS. As in [3], we use multi-layered perceptrons (MLP) [15] in place of ternary strings.

Firstly, the environmental representation was altered - the binary string normally used to represent a given state S is replaced with a real-valued counterpart in the same way as in [2]. That is, each exclusive object type the agent could encounter is represented by a random real number within a specified range ($[0.0, 0.1]$ for free space, $[0.4, 0.5]$ for an obstacle and $[0.9, 1.0]$ for the goal state). This bounded randomness attempts to loosely emulate the sensory noise that a real robot invariably encounters - increasing the difficulty of learning the environment.

The real-valued input vector, S , is processed by each member of $[P]$ in turn. Each classifier is represented by a vector that represents the connection weights of an MLP. Each weight is initialized randomly as a uniform number in the range $[-1, 1]$. Each network is fully connected, and comprises of 8 input neurons, representing the environmental state in the 8 directions surrounding the agent, a fixed number of hidden layer neurons, which varies between experiments, and 3 output neurons. The first two output neurons represent the strength of action passed to the left and right motors of the robot respectively, and the third output neuron is a “don’t-match” neuron, that excludes the classifier from the match set if it has the highest activation of the three. This is necessary as the action of the classifier must be recalculated for each state the classifier encounters, so each classifier “sees” each input. A sigmoid function is used to constrain output values between 0 and 1. The formation of $[M]$ and $[A]$ proceed as in XCS - if the classifier does match, the outputs at the other two neurons (real numbers) are mapped to a discrete movement in one of eight compass directions. This takes place in a way similar to [2], where three ranges of discrete output are possible: $0.0 < x < 0.4$ (low), $0.4 < x < 0.6$ (medium), and $0.6 < x < 1.00$ (high). The unequal partitioning is used to counteract the insensitivity of the sigmoid function to values within the extreme reaches of its range. The combined actions of each motor translate to a discrete movement according to the two motor output strengths – (high, high) = north, (high, med) = northeast, (high, low) = east, and so on. It should be noted that the final two motor pairings – (low, medium) and (low, low) both produce a move to the northwest. Covering is achieved by repeatedly generating random MLPs with a fixed number of hidden layer neurons until the MLP’s action matches the desired output for a given input state. After each matching classifier’s action is determined an action selection policy is invoked and all classifiers that advocate the chosen action form $[A]$. If the goal state is found, reward is distributed as in XCS and the task is reset.

GA crossover is removed, due to the potential competing conventions problem and the difficulty in crossing over variable-length representations. In the non-adaptive case, mutation occurs with probability μ per allele, and alters a weight by a uniform number $\pm 0.0 - 0.1$. Two further changes are employed to increase the efficiency of the system. A mechanism known as teletransportation[12] is enforced on both explore and exploit trials, to ensure that the agent is exposed more evenly to different areas of the environment. Teletransportation imposes a timeout on the system, resetting the trial if the agent has not reached the goal state after 50 discrete movements. Additionally, an explore trial is based on roulette wheel selection rather than random action selection, to discourage time-wasting movements by the

agent since we envisage using the system with a real robot platform in the near future [17].

4. EXPERIMENTATION

4.1 naN-XCS

Each experiment consists of 50,000 trials, each consisting of one exploration cycle and one exploitation cycle. An additional knowledge test is also undertaken after these two. Following notation seen in [20], parameters used are: $N=2000$ for Woods1, $N=3000$ for Maze4, $\beta=0.2$, $\gamma=0.71$, $\epsilon_0=10$, $v=5$, $\theta_{GA}=25$, $\sigma=0.1$. Each experiment is repeated 10 times, and the results are averaged for all parameters. To allow the non-adaptive experiments the best chance to solve each maze, hidden layer sizes were selected based on preliminary investigations into the maximum (9) and average (2 for Woods1, 3 for Maze4) hidden layer sizes used to successfully solve each maze. Similarly, the values of μ were taken from self-adaptive experimental trials, and are presented in Table 1.

Table 1. μ values for non-adaptive experiments

Experiment	μ value
naN-XCS 9 neurons, Woods1	0.03
naN-XCS 9 neurons, Maze4	0.05
naN-XCS 2 neurons, Woods1	0.06
naN-XCS 3 neurons, Maze4	0.08

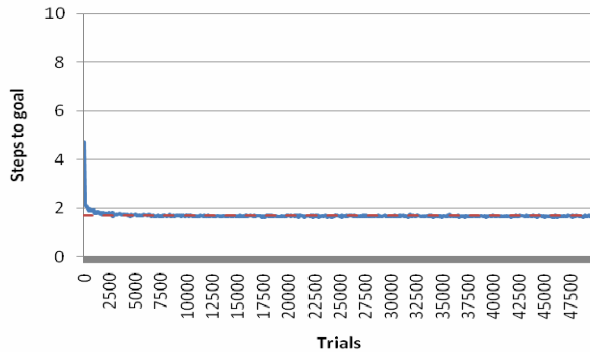


Figure 3. naN-XCS 9-neuron Woods1 performance

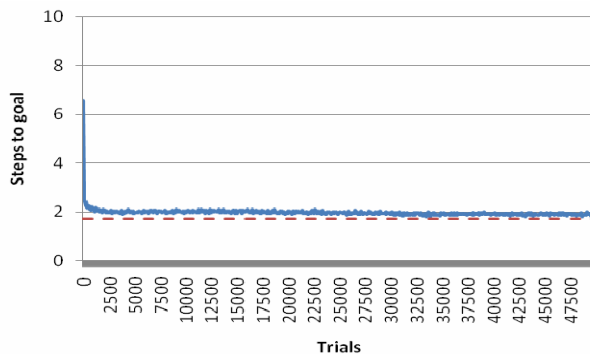


Figure 4. naN-XCS 2-neuron Woods1 performance

Figure 3 shows a steep decline towards the optimal step-to-goal value in Woods 1 for the basic N-XCS, followed by a more gradual descent until the solution stabilizes at around 7500 trials. An overabundance of neurons and an experimentally-determined μ parameter ensure that the maze is solvable, with all runs reaching optimality. However, the solution obtained for a network of two hidden layer neurons (Figure 4) is suboptimal throughout. This suggests that additional mechanisms are required to solve Woods1 with two hidden layer neurons, as only five of the ten runs reached optimality. Figures 5 and 6 show that, with an experimentally-determined μ value and a large hidden layer network size, a neural XCS with the selected parameters cannot solve Maze4. In each case, only a small number of the experiments reached optimality.

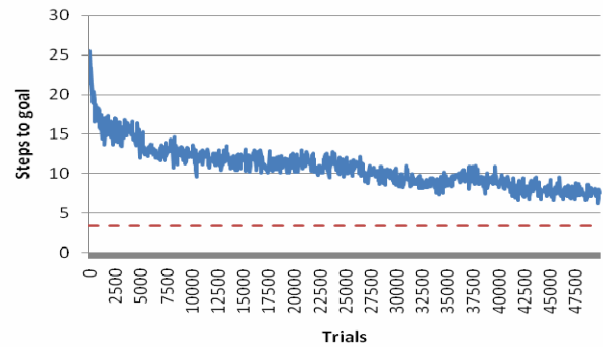


Figure 5. naN-XCS 9-neuron Maze4 performance

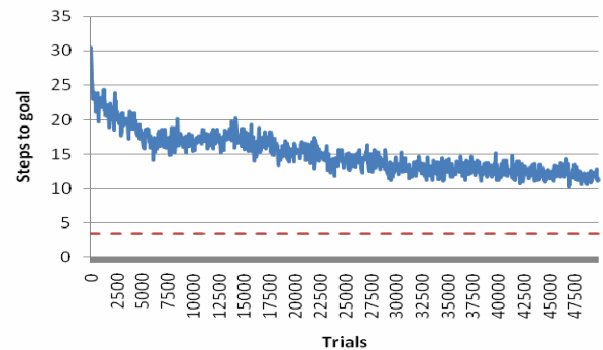
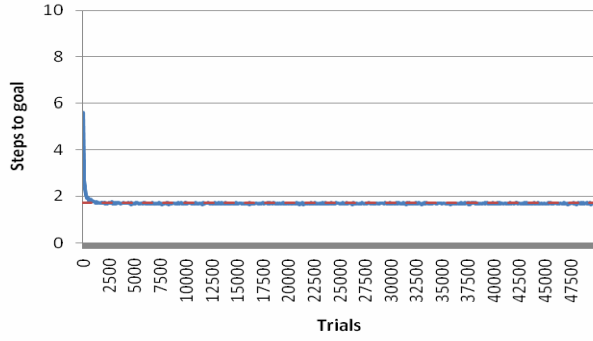


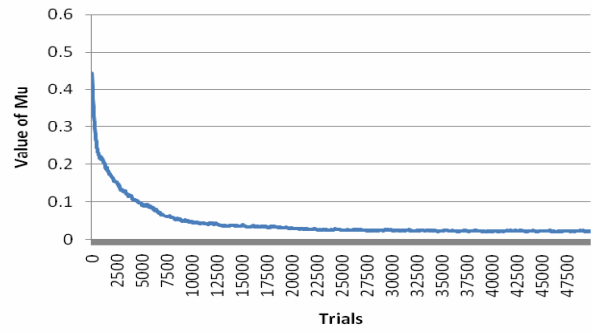
Figure 6. naN-XCS 3-neuron Maze4 performance

4.2 Self-Adaptive Mutation: saN-XCS

Bull et al. [4] describe how self-adaptive mutation can be used to dynamically control the amount of genetic search (the frequency of mutation events) taking place within a given niche. This provides stability to parts of the problem space that are already “solved” as the mutation rate for a niche is typically directly proportional to its distance from the goal state during learning; generalization learning, along with the value function learning, occurs faster nearer the goal state. Self-adaptive mutation is here applied as in [11], where the μ value of each classifier is initialized uniformly randomly in the range [0,1]. During a GA cycle, a parent’s μ value is modified in the following way before being copied to its offspring: $\mu \leftarrow \mu * e^{M(0,1)}$, clipping the result in the range [0,1] (see [16] for the effect of this update). The offspring then applies its own μ to itself before being inserted into the population.

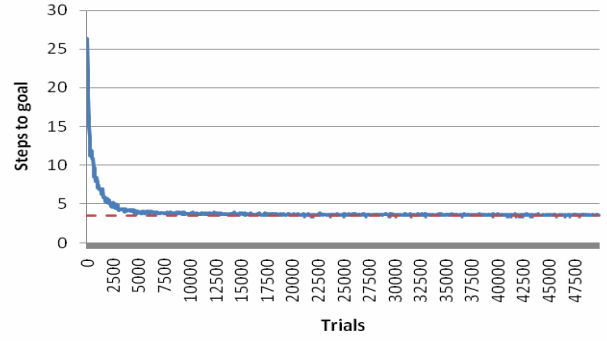


(a)

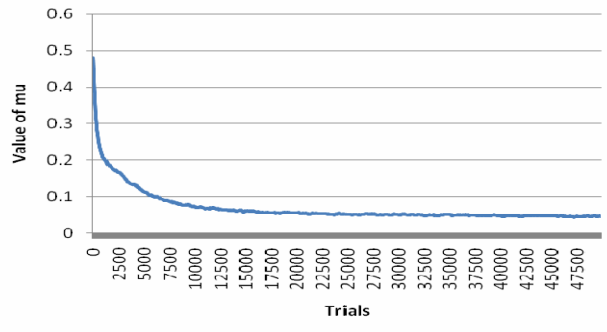


(b)

Figure 7. saN-XCS 9 neuron (a) performance and (b) μ value in Woods1

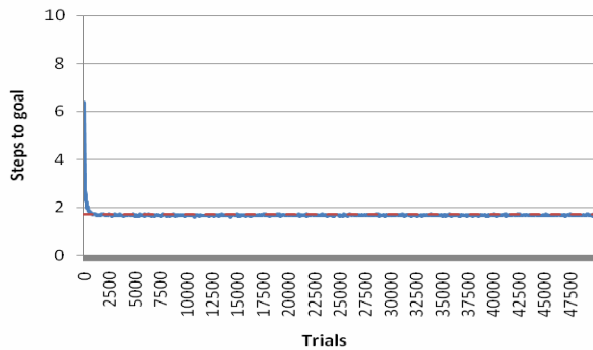


(a)

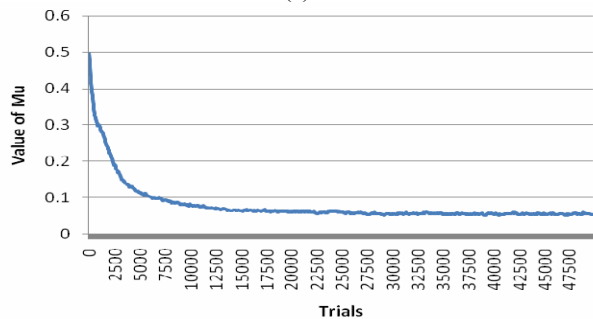


(b)

Figure 9. saN-XCS 9 neuron (a) performance and (b) μ value in Maze4

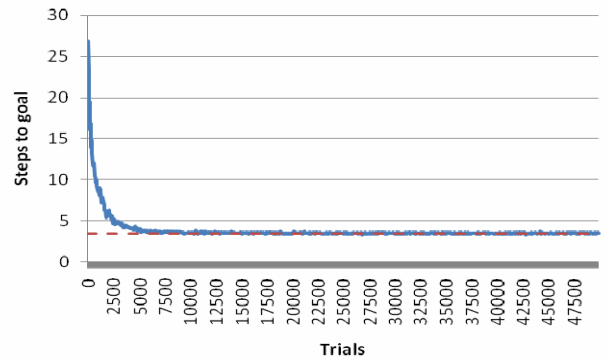


(a)

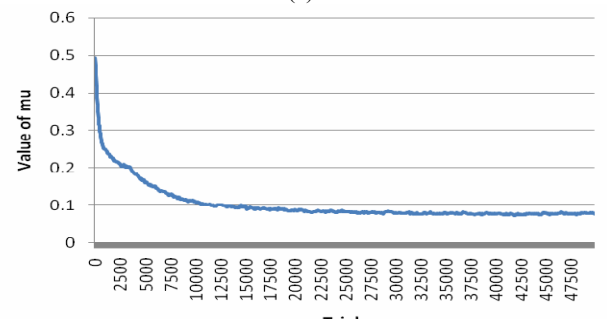


(b)

Figure 8. saN-XCS 2 neuron (a) performance and (b) μ value in Woods1



(a)



(b)

Figure 10. saN-XCS 3 neuron (a) performance and (b) μ value in Maze4

Figure 7 confirms the previous result for naN-XCS in figure 3 wherein the system with 9 hidden layer nodes is capable of optimal performance. Figure 8 shows that a network limited to two hidden layer neurons can reach optimality in Woods1, thanks to the use of self-adaptive mutation rates (contrast with the static mutation rate in figure 4). Figures 9(a) and 10(a) demonstrate that, in a more complex maze environment, self-adaptive mutation allows optimal solutions to be found for the two fixed architectures used, in contrast to the results in figures 5 and 6. The difference in final μ values seen in figures 9(b) and 10(b) can be attributed to the differing number of genes per classifier (9 hidden layer nodes vs. 3 hidden layer nodes); the self-adaptation process is context sensitive.

Examination of optimal solutions confirms results reported in [3] whereby rules evolve which match all or many of the states at a given prediction level. Thus rules generate different actions based upon the input.

4.3 Self-Adaptive Constructivism: ncN-XCS

In principle, NC allows optimal solutions to be found without over-fitting to the problem space by including an excess of hidden layer neurons. As shown above, using a static hidden layer size for the solution may cause either under- or over-fitting representations of too few or too many neurons respectively. Under NC networks can evolve to match the complexity of the environment (subspaces of the problem space) they are presented with [2].

The use of evolutionary computing techniques to allow for the emergence of appropriate complexity in neural networks was first introduced by Harvey et al. [8]. An evolutionary gradualism mechanism was used such that the length of the genotypes could increase to an appropriate size over time; extra nodes could be added to the network through a mutation-like operator during reproduction. Implementation of NC in this system is based on that described by Hurst and Bull [11]. Each rule has a varying number of hidden layer neurons (initially 1, and always ≥ 1), with additional neurons being added or removed from the hidden layer depending on the constructivism element of the system. Constructivism takes place during discovery, after mutation. Two new self-adaptive parameters, ψ and ω , are added. Here, ψ represents the probability of performing a constructivism event and ω is the probability of adding a neuron; removal occurs with probability $1 - \omega$. As with self-adaptive mutation, both are initially randomly generated uniformly in the range $[0,1]$, and offspring classifiers have their parents' ψ and ω values multiplied by $e^{N(0,1)}$ during reproduction, with the results clipped to the range $[0,1]$. Much like self-adaptive mutation, this has the potential to give a gradient to the amount of constructivism that takes place within any given niche, with the effect of keeping optimal/fit solutions stable whilst altering non-optimal niches more frequently until they reach optimality.

Figure 11(a) shows that ncN-XCS reaches optimality within 2500 trials. A high initial step-to-goal value can be observed as networks of sufficient complexity must be grown from one neuron (Figure 11(b)), and a steep gradient of descent is encountered once networks are evolved to handle that complexity. The process of constructivism adds further variable interaction, and especially early in the experiment this can be observed in the

unsteady gradient in figure 11(c), as mutation rates interact with the addition of new neurons.

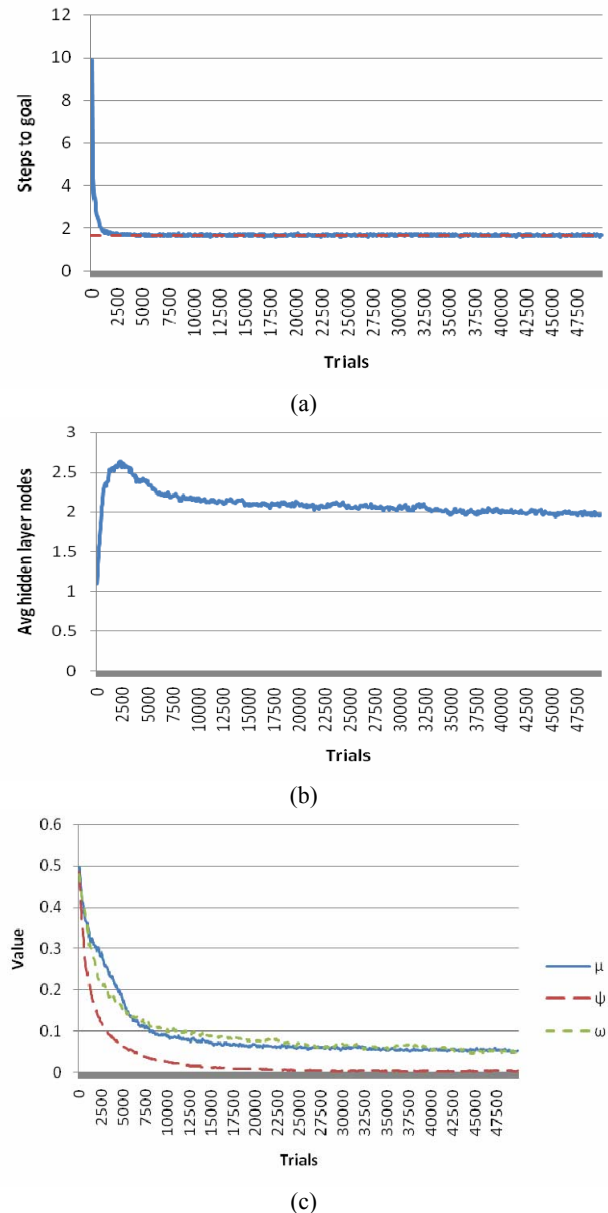


Figure 11. ncN-XCS (a) performance, (b) average hidden layer size and (c) self-adaptive parameters in Woods1

Figure 12(a) shows that the system is capable of performing optimally in Maze4 as all runs eventually reach optimality. There is a correlation at around 20,000 trials between the system reaching optimal performance, the value of μ levelling off (Figure 12(c)), and the average number of hidden layer nodes per network reaching stability (Figure 12(b)).

Note the the average number of hidden layer nodes are different in the two cases, as are the dynamic and final values of the search parameters. Further, as reported in [2] and [3], examination of solutions finds rules with different numbers of hidden layer nodes matching in different parts of a given maze.

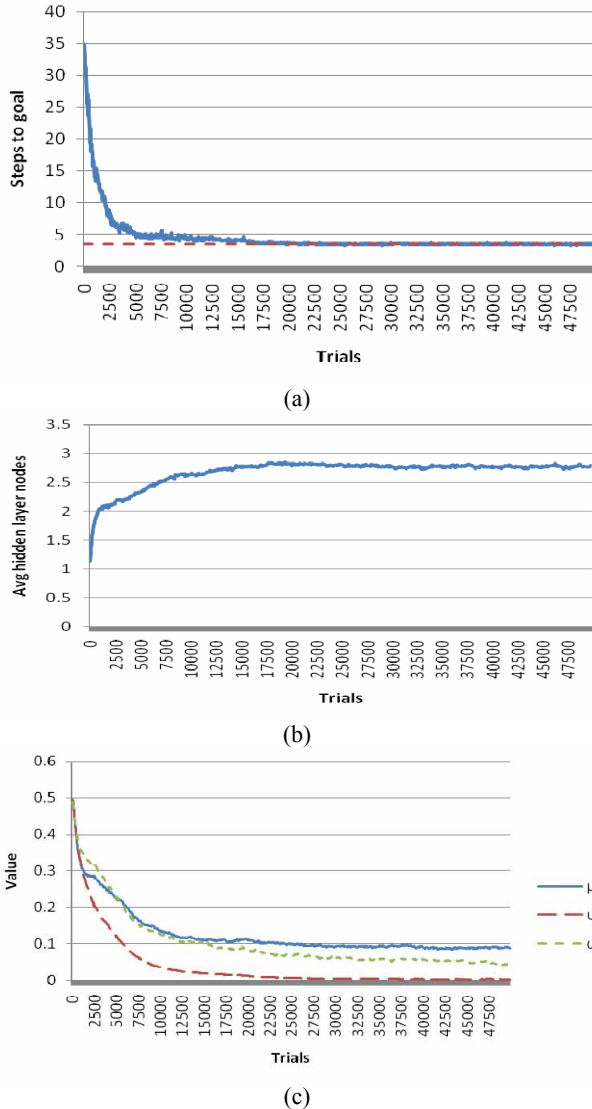


Figure 12. ncN-XCS (a) performance, (b) average hidden layer size and (c) self-adaptive parameters in Maze4

5. STATISTICAL SIGNIFICANCE

As noted in Section 4.1, after each explore-exploit cycle, an additional knowledge test trial is held with the agent always starting in the closest available location to the top-left hand corner of the maze, and the steps-to-goal count recorded. Under the standard maze scenario used above, and in the LCS literature, it is not possible to perform standard statistical tests for significant differences in performance as performance is plotted as a 50-point moving average due to the random start location. Using an extra exploit trial from a fixed position eases statistical comparison.

As each modification is added to the system (MLP \rightarrow self adaptive, self-adaptive \rightarrow self-adaptive & constructive), the average stability of a system's performance is ascertained. Stability is defined as follows: A solution can be said to be stable if, for each of 50 consecutive knowledge test trials - interspersed between standard explore and exploit trials - from a constant

location in the maze, the solution always finds the optimal path to the goal. The first trial at which each run of a system reaches stability is recorded, and this set of 10 numbers is compared to the sets produced by the other variants of the system using a standard t-test. The outcome is therefore a probability of whether the two sets of numbers belong to the same overall distribution, ($p\text{-value} > 0.01$), or different distributions ($p\text{-value} \leq 0.01$). If the latter is the case, the two sets of numbers are said to be statistically significantly different. Only systems that reached optimality during experimentation are compared. Averages are provided as a point of reference.

Table 2. T-test results on Woods1 N-XCS variants

Systems compared	Different?	System 1 Avg	System 2 Avg	P-value
naN-XCS 9 node	No	169	368	0.02
saN-XCS 9 node				
saN-XCS 2 node	Yes	325	1091	0.0001
ncN-XCS				
saN-XCS 9 node	Yes	368	1091	0.0003
ncN-XCS				

The results in Table 2 show that self-adaptation does not significantly affect the performance of the system in reaching stability when sufficient hidden layer nodes are present (although the p-value of 0.02 is close to the boundary), whereas the addition of constructivism gives the system a performance overhead. This can be attributed to the necessity of the NC system to evolve networks of sufficient complexity before a stable solution can be found.

Table 3. T-test results on Maze4 N-XCS variants

Systems compared	Different?	System 1 Avg	System 2 Avg	P-value
saN-XCS 3 node	No	13770	18199	0.16
ncN-XCS				
saN-XCS 9 node	No	13955	18199	0.08
ncN-XCS				

As neither non-adaptive system reached stability, they cannot be compared for Maze 4. Table 3 shows no statistically significant difference between saN-XCS and ncN-XCS with regards to reaching optimality in the Maze4 environment. This can be explained as follows: the overhead involved in growing networks of sufficient complexity is larger in Maze4 than in Woods1 as the environment is more complex. However, the flexible representations allowed by NC allow the system to become stable more quickly once this complexity has been reached. This suggests that, in more complex environments, the benefits of representations afforded by NC may offset the computational start-up cost of evolving the networks in the first place.

6. A NEURAL XCSF

In XCSF [20], a classifier's prediction is computed. This attempts to alleviate a drawback of XCS; that a classifier's prediction is constant in the entire problem subspace covered by its condition,

which can limit generalization capabilities. Utilizing XCSF may lead to the discovery of classifiers within the population that generalize not only within a payoff level, but also between payoff levels. Let us call this system N-XCSF.

At the start of each experiment, classifiers are initialized as in XCS, except that each classifier is augmented by a weight vector, w . This vector has one element for each input (8 in this case), plus an additional element w_0 which corresponds to x_0 , a constant input that is set as a parameter of XCSF. Each vector element is initialized as 0. At each time step, XCSF builds a match set as normal. Note that most implementations alter the traditional ternary environmental inputs to real numbers but we are already using real-valued inputs so need no modifications. An action is chosen via the formation of a prediction array. Each classifier prediction ($cl.p$) is calculated as a product of the environmental input (or state, st) and the weight vector (w) associated with each classifier, specifically:

$$cl.p(st) = cl.w_0 * x_0 + \sum_{i>0} cl.w_i * st(i)$$

These predictions are summed to form the prediction array as a fitness-weighted average of all classifiers in the match set that specify a given action. The prediction array is then used as in regular XCS to decide on an action to take (in our version, this is deterministic during an exploit trial and roulette during an explore trial). The action is then performed and reward returned from the environment. During reinforcement, the weight vector of each classifier in the action set is updated using a version of the delta rule, rather than updating the classifiers' prediction value. Here, the vector x is the state st augmented by the parameter x_0 .

$$\Delta w_i = \frac{\eta}{|x_{t-1}|^2} (r - cl.p(s_{t-1})) x_{t-1}(i)$$

Each weight is then updated.

$$cl.w_i \leftarrow cl.w_i + \Delta w_i$$

Finally, prediction error is calculated.

$$\varepsilon \leftarrow \varepsilon + \beta(|r - cl.p(s_{t-1})| - \varepsilon)$$

Discovery proceeds as in XCS. During reproduction, the weight vector of a parent is copied unmodified to its offspring, with self-adaptation applied to μ , ψ and ω as normal. All parameters were as before except the population size $N=7000$ in Maze4, $N=6000$ in Woods1, the experimentally-determined x_0 parameter is set to 10 and the correction rate η to 0.2. These parameters are based on the work of [20]. Each classifier begins with or has a hidden layer size of 2, as a 1-neuron N-XCSF of any version failed to reach optimality reliably. Figure 14(a) shows that ncN-XCSF is capable of optimal performance in the real-valued Maze4. Observation of the action sets produced indicate that ncN-XCSF has the potential to evolve classifiers that generalize between different payoff levels, resulting in a more compact solution and increased generalization capability. Figure 14(b) shows a decline

to approximately 1.5 neurons per classifier, suggesting that the inclusion of computed prediction allows less complex neural representations to perform optimally. However, we note that initialization with one node in the hidden layer for Maze 4, as used above, was found to give sub-optimal performance in one out of ten runs (not shown) with the same parameters. Current work is exploring why this was the case, although it is interesting to note that the constructivism process is able to prune solutions as well as grow them. The results of t-tests between ncN-XCSF and other variants in Maze4 (Table 4) show that there is no statistically significant difference between the compared systems with regards to reaching stability.

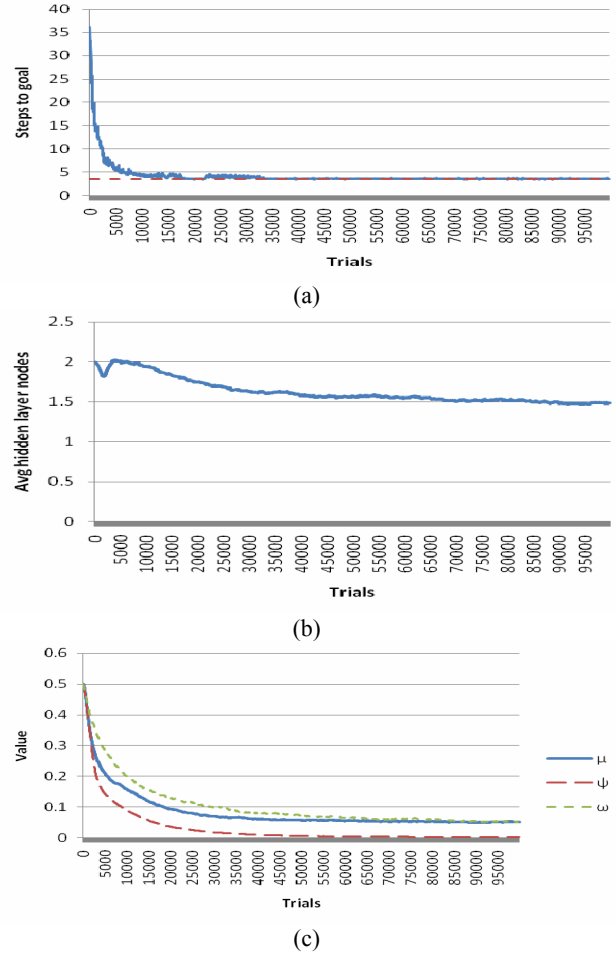


Figure 14. ncN-XCSF (a) performance, (b) average hidden layer size and (c) self-adaptive parameters in Maze4.

Table 4. T-Test results of N-XCSF in Maze4

Systems compared	Different?	System 1 Avg	System 2 Avg	P-value
ncN-XCS	No	18199	9615	0.02
ncN-XCSF				
saN-XCSF 2 node	No	8228	9615	0.51
ncN-XCSF				

With respect to steps-to-goal, ncN-XCSF and ncN-XCS initially reach optimality in comparable time. However Figure 14(a) shows a slight “hump” at around 25,000 trials which was caused by one of the ten experiments briefly becoming sub-optimal, and because of this, comparison to Figure 12(a) shows that ncN-XCSF takes longer to reach an optimal step-to-goal figure than ncN-XCS. This can be explained by the complexity of interactions between self-adaptive constructivism and prediction computation, and improving this interaction is a future aim of this work. To our knowledge, this is the first time XCSF has been used with noise included in the inputs and this may be a partial explanation; the sensitivity of the piece-wise linear approximations to noise is empirically unknown at this stage. The inclusion of NC within the XCSF framework can be seen to have a performance impact on the system. For example, Table 4 shows over a 50% probability that ncN-XCSF and saN-XCSF come from the same distribution, but a 98% certainty that ncN-XCS and ncN-XCSF are from different distributions (p-value 0.02). This can be attributed to the performance overhead involved in constructivism being offset/superseded by the benefits of computed prediction

7. CONCLUSIONS

We have shown that a self-adaptive neural XCS employing constructivism can perform optimally in noisily-encoded real-valued versions of two well-known simulated maze environments. The system evolves a population of MLPs to cover the problem space, the result being a complete payoff map of the entire problem space, where one MLP can cover a large homogenous region, i.e., where the payoff landscape is identical. Further, using the prediction computation of XCSF, we have seen that one MLP can cover several disparate regions of the problem space, typically where the action required is identical, but the payoff levels are different. We are now moving this work to a real robot platform.

8. REFERENCES

- [1] 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. San Mateo, CA: Morgan Kaufmann, pp11-18.
- [2] Bull, L. 2002. On Using Constructivism in Neural Classifier Systems. In Merelo, J, Adamidis, P., Beyer, H-G., Fernandez-Villacanas, J-L., & Schwefel, H-P. (Eds.) Parallel Problem Solving from Nature – PPSN VII. Springer Verlag, pp558-567.
- [3] Bull, L. & O’Hara, T. 2002. Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems. In W.B. Langdon, E.Cantu-Paz, K. Mathias, R.Roy, D.Davis, R.Poli, K. Balakrishnan, V. Hanavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F.Miller, E.Burke & N. Jonoska (Eds.) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann. pp905-911.
- [4] Bull, L., Hurst, J., & Tomlinson, A. 2000. Self-Adaptive Mutation in Classifier System Controllers. In J-A. Meyer, A. Berthoz, D. Floreano, H. Roitblatt & S.W. Wilson (Eds.) From Animals to Animats 6 – The Sixth International Conference on the Simulation of Adaptive Behaviour, MIT Press.
- [5] Butz, M. V., & Wilson, S. W. 2001. An Algorithmic Description of XCS. In Lanzi, P. L., Stolzmann, W., and S. W. Wilson (Eds.), *Advances in Learning Classifier Systems*, LNAI 1996, pp. 253-272. Berlin: Springer-Verlag
- [6] Cordón, O., Herrera, F., Hoffmann, F. & Magdalena, L. 2001. *Genetic Fuzzy Systems. Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. World Scientific.
- [7] Edelman, G. 1987. *Neural Darwinism: The Theory of Neuronal Group Selection*. New York: Basic Books.
- [8] Harvey, I., Husbands, P. & Cliff, D. 1994. *Seeing the Light: Artificial Evolution, Real Vision*. In D. Cliff, P. Husbands, J-A. Meyer & S.W. Wilson (eds) *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behaviour*. Cambridge, MA: MIT Press, pp392-401.
- [9] Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.
- [10] Holland, J.H. 1976. *Adaptation*. In R. Rosen & F.M. Snell (Eds.) *Progress in Theoretical Biology 4*. New York: Academic Press, pp263-293.
- [11] Hurst, J. & Bull, L. 2006. A Neural Learning Classifier System with Self-Adaptive Constructivism for Mobile Robot Control. *Artificial Life 12(3)*: 353 - 380
- [12] Lanzi, P.L. 1999. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation 7(2)*: 125-149
- [13] Lanzi, P.L. & Loiacono, D. 2007. Classifier systems that compute action mappings. In *GECCO '07: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA. ACM Press. pp1822-1829.
- [14] Quartz, S.R. & Sejnowski, T.J. 1997. The Neural Basis of Cognitive Development: A Constructionist Manifesto. *Behavioural and Brain Sciences 20(4)*: 537-596.
- [15] Rumelhart, D.E. & McClelland, J.L. 1986. *Parallel Distributed Processing*. Cambridge, MA: MIT Press
- [16] Schwefel, H-P. 1981. *Numerical Optimization of Computer Models*. Wiley, Chichester,
- [17] Studley, M. & Bull, L. 2006. Using the XCS Classifier System for Multi-objective Reinforcement Learning Problems. *Artificial Life 13(1)*: 69-86.
- [18] Wilson, S.W. 1994. ZCS: A Zeroth-level Classifier System. *Evolutionary Computation 2(1)*:1-18.
- [19] Wilson, S.W. 1995. Classifier Fitness Based on Accuracy. *Evolutionary Computation, 3(2)*:149-175.
- [20] Wilson, S.W. 2001. Function Approximation with a Classifier System. In Spector, L., D., G. E., Wu, A., Langdon, W.B., Voight, H. M., and Gen, M., (Eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 01)* Morgan Kaufmann. pp 974-981
- [21] Wilson, S.W. 2007. Three architectures for continuous action Learning Classifier Systems. *International Workshops, IW LCS 2003-2005, Revised Selected Papers*. In T. Kovacs, X. Llorà, K. Takadama, P. L. Lanzi, W. Stolzmann, S. W. Wilson (Eds.) *Lecture Notes in Artificial Intelligence (LNAI-4399)*, Berlin, Springer-Verlag. pp. 239-257