On the Effects of Node Duplication and Connection-Oriented Constructivism in Neural XCSF

Gerard David Howard
Department of Computer Science
University of the West of England
Frenchay, Bristol, UK
+44 (0)117 965 6261
gerard2.howard@uwe.ac.uk

Larry Bull
Department of Computer Science
University of the West of England
Frenchay, Bristol, UK
+44 (0)117 965 6261
larry.bull@uwe.ac.uk

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 behavior. Various network growth/regression mechanisms are implemented and their performances compared. 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.

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) [17] 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 XCSF [25]. In particular, we explore the success of extensions to the XCSF-based neural LCS, N-XCSF[11], including node duplication and a connection-oriented neural constructivism scheme, on a real-valued version of a well-known maze task. The main differences between XCSF and N-XCSF are that, in N-XCSF, a neural network is used in place of the condition-action pair, and self-adaptive mutation parameters are included.

We shall refer to the four systems presented using the following nomenclature: neural constructive XCSF (ncN-XCSF), neural constructive XCSF with node duplicaton (ncN-XCSFd), and neural constructive XCSF with duplication and connection constructivism (ncN-XCSFdc / ncN-XCSFdce / ncN-XCSFdcτ). To our knowledge, this is the first comparative study of the above constructivism mechanisms that uses N-XCSF as a basis. The paper is ordered as follows: the next section provides a brief overview of related work. Section 3 describes the implementations of the various systems. Section 4 presents the results of ncN-XCSF in solving a well-known maze environment, and compares the results with the same experiments attempted by ncN-XCSFd, and variants of ncN-XCSFdc. Section 5 draws conclusions from the results and suggests directions for future work.

2. RELATED WORK
The use of constructivism within neural learning classifier systems was first described by Bull [3], using Wilson's ZCS [23] as a basis. Hurst and Bull [12] 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. An overview of the effects of various methods of neural constructivism applied to the traditional evolution of single neural networks can be found in [20].

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

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

overview); Lisp S-Expressions [1]; and parameterized functions [14, 26].

3. IMPLEMENTATION

3.1 Maze environment

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 environment used in this paper is the benchmark Maze4[15]. 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 Maze4 the optimal figure is 3.5; Figure 1 shows the layout. Here, "O" represents an obstacle, "*" an empty space and "G" the goal state.

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 1. The Maze4 Environment

3.2 A Neural XCSF

In XCSF [25], a classifier's prediction (that is, the reward a classifier *expects* to gain from executing its' action based on the current input state), is computed. This attempts to alleviate a drawback of XCS (e.g., [24, 5]); 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.

Adapting the work of Bull and O`Hara [2], a number of changes were made to the standard XCSF algorithm to accommodate the use of artificial neural network rules. As in [3], we use multi-layered perceptrons (MLPs) [19] 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 [3]. 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 whilst also 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 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 re-calculated for each state the classifier encounters, so each classifier "sees" each input. A sigmoid function is used to constrain the MLP output node values between 0 and 1.

The outputs at the other two neurons (real numbers) are mapped to a discrete movement in one of eight compass directions. This takes place similarly to [3], 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 single hidden layer neuron (unless otherwise stated), 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.

At the start of each experiment, the classifiers' prediction components (linear approximators) are initialized with 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, [M], that consists of all classifiers that match the current input. An action set is generated 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 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. All other updating of the vector w is as described in [25].

GA crossover is removed, due to the potential competing conventions problem and the difficulties associated with crossing variable-length representations (although it is possible, e.g., see [8, 13]).

Two further changes are employed to increase the efficiency of the system. A mechanism known as teletransportation[15] 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, as noted, as explore trial is based on roulette wheel selection rather than random action selection, to discourage time-wasting movements by the agent; we envisage using the system on a real robot in the near future [22].

We apply self-adaptation as in [4], 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 [12], 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: $\mu \leftarrow \mu * e^{N(0,1)}$, with the result being clipped to the range [0,1]. The offspring then applies its own μ to itself before being inserted into the population.

In each experiment (unless stated to be otherwise), each classifier begins with one fully-connected hidden layer neuron; appropriate complexity is grown through constructivism at a rate determined by the self-adaptive parameters of the system.

4. EXPERIMENTATION

Each experiment consists of 100,000 trials, each consisting of one exploration cycle and one exploitation cycle. Parameters used are (using notation from[25]) N=7000, β=0.2, γ= 0.71, ε_0= 10, ν=5, θ_{GA} = 25, σ= 0.1.; additionally the experimentally-determined x_0 parameter is set to 10 and the correction rate η to 0.2. Each experiment is repeated 10 times, and the results are averaged for all parameters.

4.1 ncN-XCSF

Implementation of NC in this system is based on the work of Hurst and Bull [12]. Each rule has a varying number of hidden layer neurons (initially 1, and always > 0), 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, with removal occurring with probability 1- ω. As with self-adaptive mutation, both are initially randomly generated uniformly in the range [0,1]. Offspring classifiers have their parents' ψ and ω values multiplied by $e^{N(0,1)}$ during reproduction, and are 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. A constructivism event either adds a randomly created node or removes the last added node depending upon the satisfaction of ω. The performance of ncN-XCSF can be seen in Figure 2(a). Initially, the seen instability was attributed to a possible difficulty encountered by ncN-XCSF in growing solutions of sufficient complexity in a noisy maze environment. To test this hypothesis, we increased the starting number of hidden layer nodes to 2, so as to reduce the amount of network growth that would be necessary to reach a solution. Figure 3(a) shows that this has eliminated the problem to a degree. Figure 3(b) shows how the average number of hidden layer nodes has decreased from 2 to around 1.5. It is postulated that, if left for a longer period, the average number of nodes seen in Figures 2(b) and 3(b) would converge.

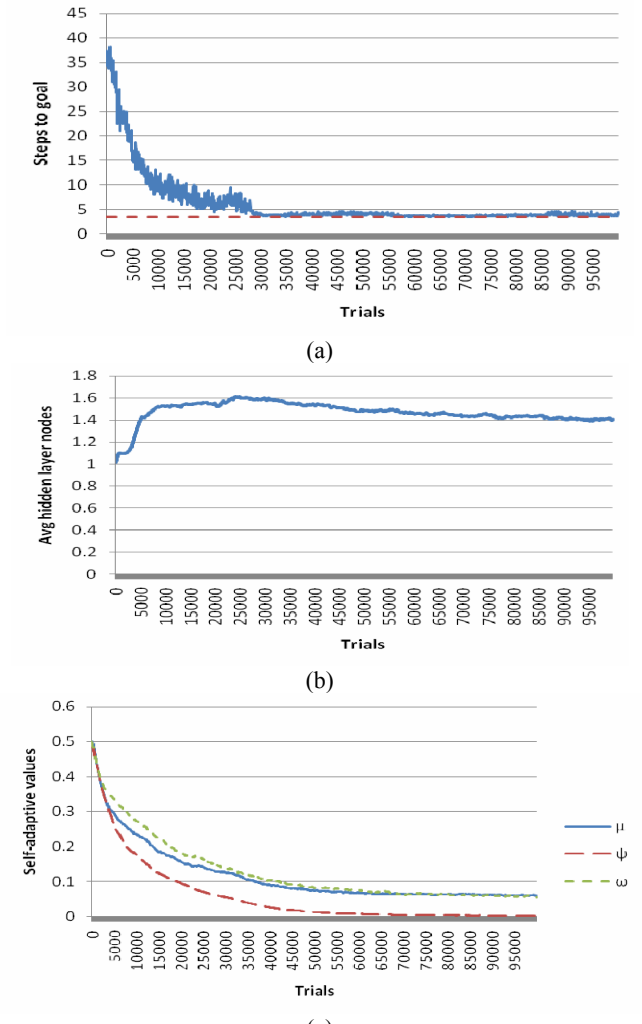
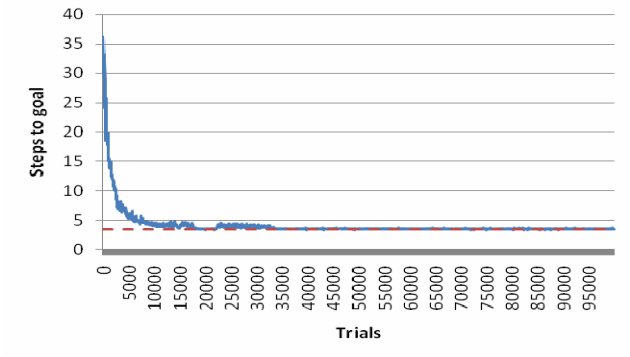
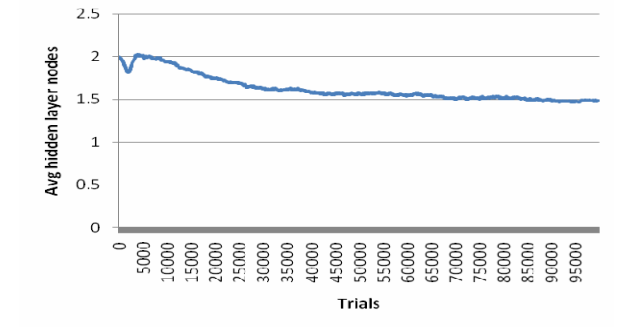


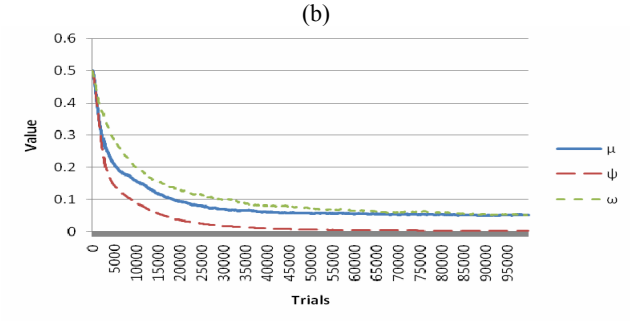
Figure 2. ncN-XCSF 1 node (a)performance (b) connected nodes (c) average self-adaptive values



(a)



(b)



(c)

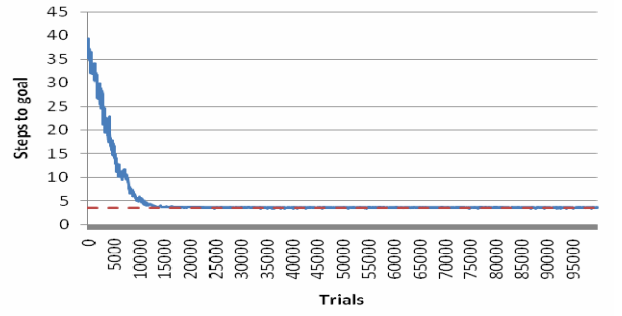
Figure 3. ncN-XCSF 2 nodes (a) performance, (b) average hidden layer size and (c) average self-adaptive values

The fact that the system is optimal for large periods of time suggests that some disruption is causing the divergence from optimality. One possible explanation is that the addition/removal of fully-connected neurons due to constructivism, alongside the prediction computation, creates some interaction of variables that is detremental to the performance of the system. That is, the impact of adding or removing a neuron is large enough in some cases to disrupt the accuracy of the prediction computation. A recent study undertaken into the evolvabiltiy of various structural mutations [20] in single networks suggests various methods to lessen disruption in general caused by constructivism. We base our implementation of potential solutions on this work.

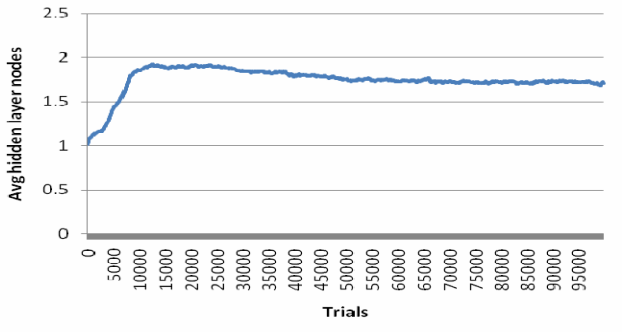
4.1.1 ncN-XCSF with Duplication (ncN-XCSFd)

Within the constructivist framework, node duplication refers to the process of copying of a pre-existing node rather than randomly generating a new nodes' connection weights during a node addition event. The biological plausibility of a duplication-based approach to constructivism is examined in [16], which highlights the importance of a cycle of duplication followed by divergence to evolution in nature. Empirical evidence of the benefits of duplication can be seen in [21], where a duplication scheme is used to improve the performance of an Evolutionary Strategy (ES) [18] to optimise multi-modal fitness functions.

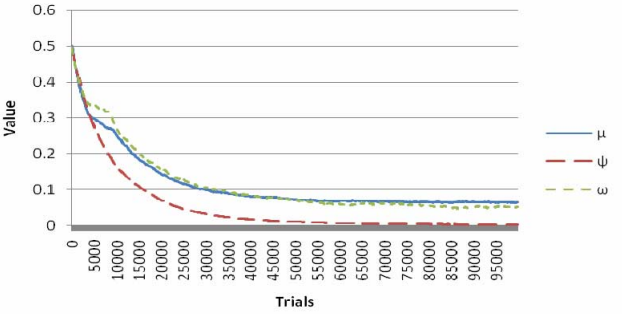
Previously, new neurons were added with random weights in the range [-1,1], which could create large changes in the functionality of the network. Now, during a constructivism event, one of the offsprings' hidden neurons is selected at random, duplicated, mutated using the classifiers' current value of μ, and added back to its' genome. In this way, much of the functionality in the new neuron is comparable to that of the old neuron; exploration of the solution space is slower but potentially less disruptive. Figure 4(a) shows that ncN-XCSFd is capable of optimal performance in the real-valued Maze4. Comparison with Figure 3(a) shows a faster descent to optimality, indicating that duplication during constructivism is lessening disruption.



(a)



(b)



(c)

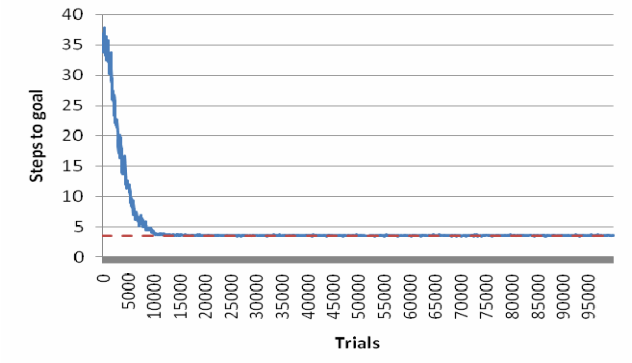
Figure 4. ncN-XCSFd (a)performance (b) connected nodes (c) average self-adaptive values

4.1.2 ncN-XCSFd with Connection-oriented Constructivism (ncN-XCSFdc)

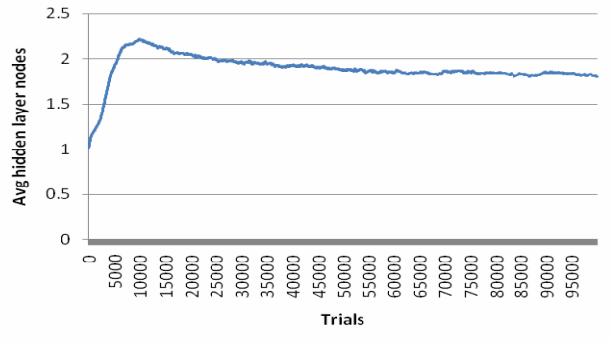
The addition of node duplication was previously used to reduce disruption during a GA cycle. Although it has been shown that duplication allows Maze4 to be solved optimally (Figure 4(a)), it is unknown whether later, more complex versions of the system will suffer from disruption that is too great for duplication to offset. With this in mind, we return to [20], which suggests that an enhanced approach may consider the use of constructivism at the level of an individual connection, rather than a fully-connected neuron. Results from the same paper show that a purely connection-based constructivism approach produces highly evolvable solutions. Further benefits include increased flexibility owing to a more granular constructivism mechanism, and the potential to evolve more compact solutions by disabling superfluous connections within a network. It should be noted however, that the comparisons presented in [20] analyse only single mutation types. We present here a hybrid system, incorporating both node constructivism and connection constructivism simultaneously.

A connection-oriented scheme is implemented as follows: each network connection is assigned a Boolean flag, which may be 0 (connection disabled) or 1 (connection enabled). Any disabled connections have their weights set to 0 so that they do not influence the result of the MLP calculation, and during a GA cycle are not viable candidates for mutation (so that their 0-value may not change unless they are enabled). Flags may be "switched" during reproduction, with a new self-adaptive parameter τ (applied in the same way as μ, ψ and ω) governing the probability that a flag is switched. If a connection is disabled then subsequently enabled, its new weight is randomly generated uniformly in the range [-1,1].

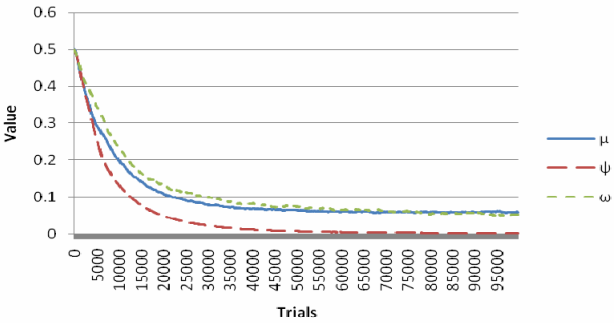
Neuron addition and removal are unchanged; when a MLP is generated (i.e., through cover) or modified (i.e., through node addition), all connection flags are set to 1 by default. It should be noted that, although connections may be enabled or disabled, they may never exceed the number of connections in the fully-connected network (new connections will not bypass the hidden layer).



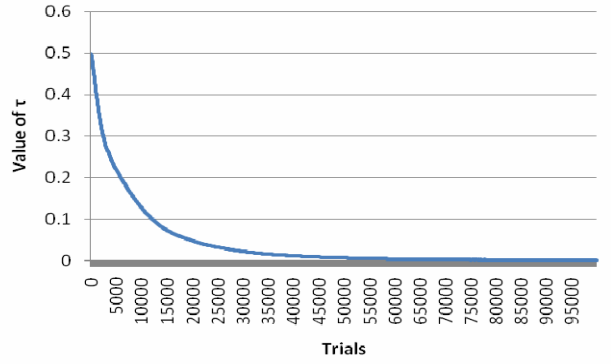
(a)



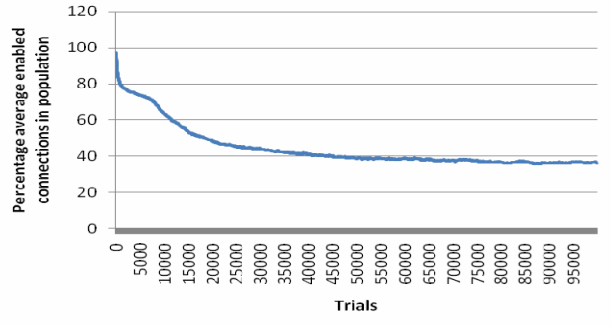
(b)



(c)



(d)



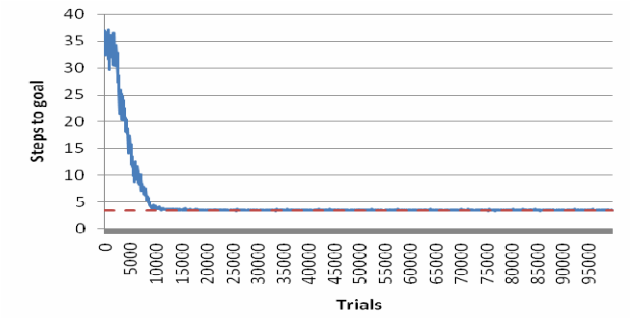
(e)

Figure 5. ncN-XCSFdc (a)performance (b) connected nodes (c) average self-adaptive values (d) value of τ and (e) average enabled connections.

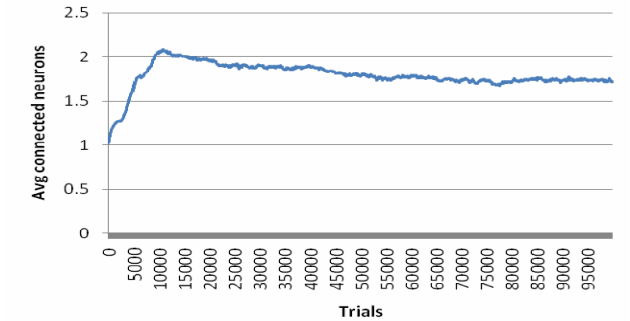
As Figure 5(a) shows, the use of connection-oriented constructivism allows the system to solve Maze4 optimally, and slightly quicker than ncN-XCSFd. The adaptation of the newest constructivism parameter, τ, can be seen in Figure 5(d), with the average percentage of enabled connections per classifier shown in Figure 5(e). It is thought that for a neural representation to function, information from all surrounding locations is needed to make an accurate decision with regards to movement in the maze (i.e. keeping a Markov problem structure). Experimental results show that almost all connections are enabled in the first layer (between input and hidden neurons), compared with fewer in the second layer (between hidden and output neurons). This has the effect of preserving the problem structure and allows a stable solution to be evolved.

4.1.2.1 Alternate methods of connection-oriented constructivism

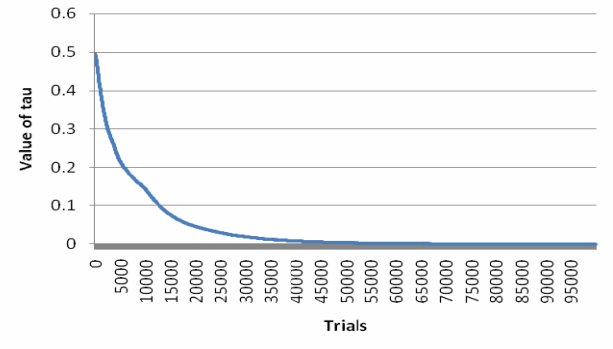
As Figure 5(e) shows, only around 40% of the available connections need to be enabled for the system to reach an optimal solution. It is therefore suggested that fully-connected node addition during neuron addition could also be detrimental to the performance of the system, compared to schemes where less connection pruning may be required. This section considers the use of two such schemes; firstly, where node constructivism adds a node whose connections are initially enabled or disabled with equal probability (ncN-XCSFdce), and secondly where the probability of a connection being initially disabled is equal to the classifiers' current value of τ (ncN-XCSFdcτ). Specifically, under ncN-XCSFdce, a flag for a new connection is enabled with a probability of 0.5 upon node creation, else it begins disabled. Similarly, ncN-XCSFdcτ flags are enabled with a probability equal to (1 − τ) upon node creation, and hence disabled with probability τ. In both systems, connection constructivism is used in two places within a GA cycle; during mutation, and during node addition.



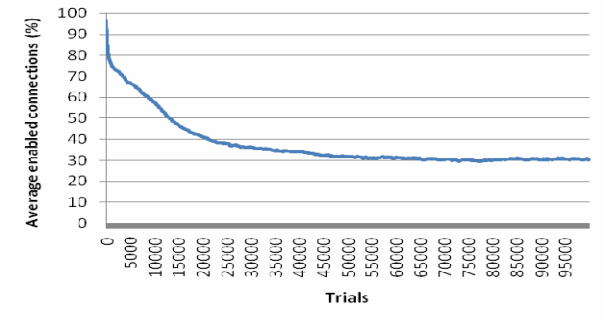
(a)



(b)



(c)

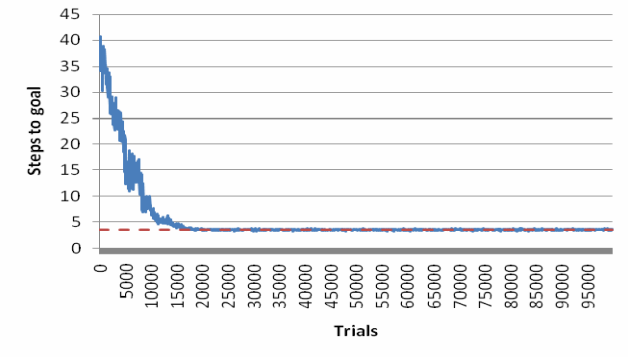


(d)

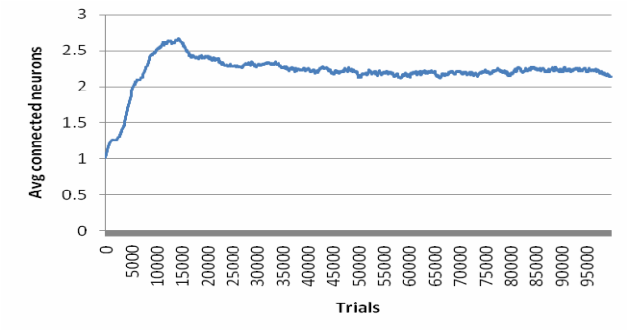
Figure 6 ncN-XCSFdce (a) Steps to goal (b) connected neurons (c) value of τ (d) average percentage enabled connections

Comparison of Figures 6(d) and 7(d) shows that ncN-XCSFdce is capable of generating significantly smaller solutions than ncN-XCSFdcτ, with around 5% fewer enabled connections utilized. Figure 6(a) shows that ncN-XCSFdce reaches stability roughly 5000 trials before ncN-XCSFdcτ, as well as using fewer neurons in its' overall solutions (~1.7 compared to ~2.2). A possible explanation for the performance disparity is that using τ to govern neuron connections during node constructivism and connection-constructivism events ties too much significance to one parameter, robbing the system of some of the versatility that a random approach can yield.

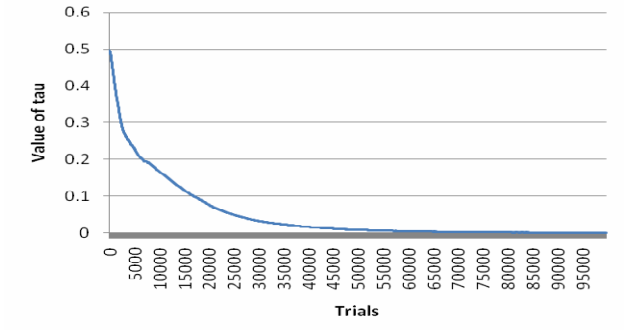
If a node constructivism event occurs when τ is a very low value, the probability of altering the neuron is so small that connection constructivism will have negligible impact on the solution (although by this point, around 55,000 trials, the solution is generally stable). Conversely, a random approach allows more flexibility at the beginning of an experiment, allowing the problem space to be more widely searched, whilst still allowing self-adaptation of τ to tailor the rate of connection constructivism during mutation. Intrinsically, the rate of connection constructivism during node addition is governed by both the probability of a node constructivism event occurring, (ψ), and the probability of adding a node, (ω), as both are prerequisites to a connection constructivism event occurring. ncN-XCSFdc is most similar to ncN-XCSFdce (compare the similarities between Figure 4(a) and 6(a), and Figure 4(b) and 6(b)). Figure 6(b) contrasts Figure 7(d), showing that ncN-XCSFdce evolves less-connected solutions (~38% compared to ~30%), and hence simpler neural representations.



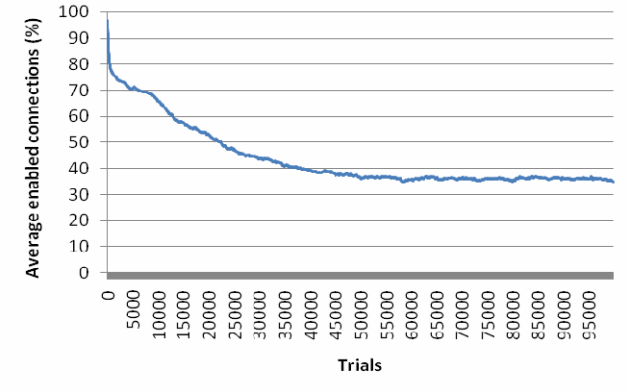
(a)



(b)



(c)



(d)

Figure 7. ncN-XCSFdcτ (a) Steps to goal (b) connected neurons (c) value of τ (d) average percentage enabled connections

5. CONCLUSIONS

We have shown that a self-adaptive and constructive neural XCSF 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 problem space, where one classifier can cover multiple homogenous regions. Furthermore, 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.

In our studies we have investigated various methods of constructivism and highlighted, from a performance standpoint, the benefits and drawbacks of each. It has been demonstrated that fully-connected neurons are not required to solve the Maze4 environment, and optimal solutions have been presented that display as little as 30% neuronal connectivity, which reduces the number of calculations necessary per classifier. We have come to the conclusion that connection-oriented schemes allow for more granular (and therefore less disruptive) network mutations than schemes based purely on fully-connected node addition/removal. We have also shown the benefits of node duplication as opposed to random node addition, especially with respect to lessening disruption in the XCSF prediction computation. Further work includes a move from discrete to continuous maze environments, and later implementing this work on a real robot platform.

6. 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. & 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.

[3] 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.

[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. 1992. Species Adaptation Genetic Algorithms: A Basis for a Continuing SAGA. In F. Varela & P. Bourgine (eds) *Proceedings of 1st European Conference on Artificial Life*. MIT Press, pp346-354

[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] Howard, D., Bull, L. & Lanzi, P-L. 2008. Self-Adaptive Constructivism in Neural XCS and XCSF. In M. Keijzer et al. (eds) GECCO-2008: Proceedings of the Genetic and Evolutionary Computation Conference. ACM Press.

[12] Hurst, J. & Bull, L. 2006. A Neural Learning Classifier System with Self-Adaptive Constructivism for Mobile Robot Control. Artificial Life 12(3): 353 - 380

[13] Hutt, B.; Warwick, K. 2007. Synapsing Variable-Length Crossover: Meaningful Crossover for Variable-Length Genomes, Evolutionary Computation, IEEE Transactions on , vol.11, no.1, pp.118-131

[14] 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.

[15] Lanzi, P.L. 1999. An Analysis of Generalization in the XCS Classifier System. Evolutionary Computation 7(2): 125-149

[16] Maynard Smith, J. & Szathmary, E. 1995. The Major Transitions in Evolution. W.H. Freeman

[17] Quartz, S.R. & Sejnowski, T.J. 1997. The Neural Basis of Cognitive Development: A Constructionist Manifesto. Behavioural and Brain Sciences 20(4): 537-596.

[18] Rechenberg, I. 1973. Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Fommann-Holzboog, Stuttgart

[19] Rumelhart, D.E. & McClelland, J.L. 1986. Parallel Distributed Processing. Cambridge, MA: MIT Press

[20] Schlessinger, E., Bentley, P. J. and Lotto, R. B. 2005. Analysing the Evolvability of Neural Network Agents through Structural Mutations. Proc. of European Conference on Artificial Life (ECAL 2005), September 5-9, 2005, Canterbury, UK.

[21] Schmitt, K. 2005. Using Gene Deletion and Duplication in Evolution Strategies. In H-G. Beyer et al. (Eds) Proceedings of the Genetic and Evolutionary Computation Conference 2005. ACM Press, pp919-920

[22] Studley, M. & Bull, L. 2006. Using the XCS Classifier System for Multi-objective Reinforcement Learning Problems. Artificial Life 13(1): 69-86.

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

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

[25] 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

[26] Wilson, S.W. 2007. Three architectures for continuous action Learning Classifier Systems. International Workshops, IWLCS 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