A Neural Learning Classifier System with
Self-Adaptive Constructivism for Mobile Robot Control

Jacob Hurst & Larry Bull

University of the West of England,
Bristol, BS16 1QY, U.K.

For artificial entities to achieve true autonomy and display complex life-like behaviour they will need to exploit appropriate adaptable learning algorithms. In this sense adaptability implies flexibility guided by the environment at any given time and an open-ended ability to learn appropriate behaviours. This paper examines the use of constructivism-inspired mechanisms within a neural learning classifier system architecture which exploits parameter self-adaptation as an approach to realise 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 learner and that the structure indicates underlying features of the task. Results are presented in simulated mazes before moving to a mobile robot platform.

Keywords: adaptation, genetic algorithm, neural network, reinforcement learning, robotics.
1. Introduction

Neural Constructivism (NC) [Quartz & Sejnowski, 1997] proposes a scenario whereby the representational features of the cortex are built through the interactions of the learning entity’s development processes and its environment. We are interested in the feasibility of a constructive approach to realize flexible learning within both simulated and real entities, in an architecture which combines neural networks, reinforcement learning and evolutionary computing. Such machine learning techniques have often been used to control autonomous entities. However, in almost all cases the entities acquire knowledge within a predefined representation scheme. Conversely, biological systems, through individually experienced sensory input and motor actions, constantly acquire new information and organize it into operational knowledge which then shapes future behaviour. Approaches which generate knowledge representations to bring new meaning to the functionality of the system are fundamental to the realization of truly autonomous entities. The aim of this research is to move toward artificial entities which exhibit such life-like qualities, based around the Learning Classifier System (LCS) [Holland, 1976] framework and Neural Constructivism.

The production of embodied intelligence requires the consideration of a number of issues including, but not limited to: the learning architecture, which must be flexible and responsive to environmental change, with automatic shifts in computational effort; and the knowledge representation, needed to provide generalization abilities over the input/output space thereby reducing the size of internal models and which must allow the inclusion of dimensions such as temporal context. In this paper we examine the suitability of a self-adaptive learning system which considers these key issues within a coherent whole - the Neural Learning Classifier System [Bull, 2002; Bull & O'Hara, 2002].

The paper is arranged as follows: the next section describes the self-adaptive Neural Learning Classifier System and in Section 3 the results of its use in a simple maze are presented. Section 4 describes the changes made to include a constructivism process and Section 5 describes the resulting behaviour of this system. The mobile robot platform is then presented with details of experimentation within stationary and non-stationary problem domains. The results from these experiments are then presented. Finally, all findings are discussed.
2. A Neural Learning Classifier System

The Neural LCS (NCS) used here is based on ZCS [Wilson, 1994], which is a simple LCS that has been shown to have the potential to perform optimally through its use of fitness sharing [Bull & Hurst, 2002]. It periodically receives an input from its environment, determines an appropriate response based on this input and performs the indicated action, usually altering the state of the environment. Desired behaviour is rewarded by providing a scalar reinforcement. Internally the system cycles through a sequence of performance, reinforcement and discovery on each discrete time-step.

The NCS rule-base consists of a population of $P$ multi-layered perceptrons (MLPs). Each rule is encoded as a string of connection weights; full connection is assumed. Weights are initialised uniformly randomly in the maximum allowed range [-1.0, 1.0]. Also associated with each rule is a fitness scalar $f$ initialised to a predetermined value $f_0$ and a mutation rate $\mu$ initialised uniformly randomly in the allowed range [0.0, 1.0].

On receipt of a sensory input, each rule has one input node per input feature, all members of the rule-base process the input in the usual manner for an MLP using a sigmoid transfer function. Each rule has an "extra" output node which signifies whether its output should be considered for the current input. If this node does not have the highest activation level, the given rule's condition "matches" the input and it is tagged as a member of the current match set [M]. An action is selected from those advocated by the rules comprising [M]. Rules propose an action by having their highest activation on a given output layer node. Action selection in NCS is performed by a simple roulette wheel selection policy based on fitness. Once an action has been selected, all rules in [M] that advocate this action are tagged as members of the action set [A] and the system executes the action.

Reinforcement in NCS is done under the implicit bucket brigade [Wilson, 1994] which is closely related to Sutton's TD(0) algorithm [Sutton, 1986] and consists of redistributing fitness between subsequent [A]. A fixed fraction ($\beta$) of the fitness of each member of [A] at each time-step is placed in a "common bucket". A record is kept of the previous action set [A] 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.
If a reward is received from the environment then a fixed fraction (β) of this value is distributed evenly amongst the members of [A]. More formally (after [Wilson, 1994]):

\[ S_{[A]} \leftarrow β \cdot r_{imm} + γS_{[A]}' \]

where \( S_{[A]} \) is the fitness of the current actionset, \( S_{[A]}' \) is the fitness of the succeeding actionset, \( r_{imm} \) is the immediate external reward, and \( \leftarrow β \) is the Widrow Hoff learning procedure.

NCS employs two discovery mechanisms, a genetic algorithm (GA) [Holland, 1975] that operates over the whole rule-set (panmictically) and a covering operator. On each time-step there is a probability \( p \) of GA invocation. When called, the GA uses traditional roulette wheel selection to obtain two parent rules based on their fitness. Two offspring are produced per GA invocation. These offspring are copies of the parental rules that have been mutated at a per gene rate which is determined by themselves. That is, each rule has its own mutation rate \( μ \) which is passed to its offspring. The offspring then applies its mutation rate to itself using the following update \( μ' = μ \cdot e^{N(0,1)} \) (as in Evolutionary Strategies, see [Baeck, 1995] for an overview), before mutating the rest of the rule at the resulting rate. Upon satisfaction of the probability, genes are altered using a step size taken from a Gaussian distribution \( N(0,1) \). The parents then donate half of their fitness to their offspring who replace existing members of the population. The deleted rules are chosen using roulette wheel selection based on the reciprocal of fitness. It can be noted that without recombination, the potentially troublesome competing conventions problem [Montana & Davis, 1989] (similarly fit solutions represented by different encodings) is avoided, as is the need for a modified operator to handle variable length individuals (as experienced under the constructivism process).

If on some time-step, \([M]\) is empty or has a combined fitness of less than \( φ \) times the population average, then a covering operator is invoked. A random rule is created which matches the environmental input. The new rule is given a fitness equal to the population average and inserted into the population over writing a rule selected for deletion as before.

Typical parameters used here are: Rule-base \( (P) = 1000 \), initial rule fitness \( (f_0) = 20.0 \), learning rate \( (β) = 0.9 \), discount factor \( (γ) = 0.3 \), covering trigger \( (φ) = 0.5 \) and GA rate per time step \( (p) = 0.25 \).
Hence NCS is a reinforcement learner [Sutton & Barto, 1998] which uses evolutionary computing to design appropriate generalizations in the state-action space. These generalizations are constructed within one or more neural networks by the learning process and the environment interaction rather than by an *a priori* determination.

### 3. Simulated Maze Tasks

#### 3.1 Woods 1

![Figure 1: The Woods 1 environment](image)

Figure 1 shows the well-known Woods 1 [Wilson, 1994] maze task which is a two dimensional rectilinear 5x5 toroidal grid. Sixteen cells are blank, eight contain trees and one contains food. The NCS is used to develop the controller of a simulated robot which must traverse the map 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 robot 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 robot randomly relocated. This very simple maze is here used to demonstrate and analyze the features of the NCS, i.e., in an easily understood context, before moving to the real mobile robot environment.

On each time step the robot 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 robot represented by the first bit-pair, and then proceeding clockwise around it.
The trial is repeated 30,000 times and a record is kept of a moving average (over the previous 50 trials, after Wilson, 1994) of how many steps it takes for the NCS robot to move into a food cell on each trial. If it moved randomly Wilson calculates performance at 27 steps per trial, whilst the optimum is 1.7 steps. For the last 2000 trials the GA is switched off and a deterministic action selection scheme is used whereby the action with largest total fitness in $M$ is picked (after Bull & Hurst, 2002). All results presented are the average of ten runs.

Figure 2: a) NCS performance in Woods1.

Figure 2: b) Movement of self-adaptive mutation rate in Woods1.

Figure 2(a) shows how NCS is able to solve the maze optimally. Here each rule has sixteen input, four hidden and nine output nodes. The parameters used were as given in Section 2. It can be seen that NCS takes around three steps to food during learning/normal usage, before giving optimal performance under the deterministic mode. Examination of the resulting rule-bases shows two basic scenarios can occur. In the first, rules emerge which handle locations with the same external payoff value, particularly for the cells which are one step from the food. That is, generalizations are seen in both the input space and action space. In the second case rules emerge for each action. For example, a single rule is used for the three locations which require a southeast move in the
top left corner of Woods 1 (as shown in Figure 1). Another rule emerges to provide the move east required for the remaining location in that area.

Figure 2(b) shows the behaviour of the average mutation rate of the population. It can be seen that the rate drops during the course of learning as a solution is converged upon. Using the traditional trinary rule encoding, Bull et al. [2000] describe how the incorporation of self-adaptive mutation in LCS enables an automatic increase or decrease in the amount of exploration/learning undertaken at any time within a given niche/subset of rules. That is, learning effort is focused on unsolved tasks whilst learnt skills remained unchanged; the rate of decrease in the mutation rate for a given niche is directly proportional to the distance of that niche/location from the goal state.

3.2 Non-Stationary Woods 1

As noted in the introduction, an appropriate learning architecture for intelligent autonomous systems must be responsive to changes in the environment. Figure 3(a) shows the performance of NCS in a version of Woods 1 where the position of the goal state is moved after 15,000 trials. That is, the goal was moved from the top right corner of the "wood region" to its top left corner. It can be seen that the system experiences a drop in performance after the change before quickly compensating for the alteration. Figure 3(b) shows how the average mutation rate in the population automatically increases after the change before settling down again. Hurst and Bull [2001] have presented analysis of this general behaviour using the traditional trinary encoding of LCS, showing how a system with self-adaptive mutation experiences less of a drop in performance than one in which the mutation rate is set at an appropriate but fixed level. The same is assumed to be true here for the neural rule representation, although this has not been confirmed experimentally. All parameters were as before
4. Neural Constructivism

The Neural Constructivist [Quartz & Sejnowski, 1997] explanation for the emergence of complex 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 emphasise regressive mechanisms whereby initial neural over-connectivity is pruned based on a measure of utility [Edelman, 1987]. The scenario for constructionist 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 \textit{a priori}. The representation of the problem space is flexible and tailored by the learner's interaction with it.

Redding et al. [1993] have used heuristics to add hidden nodes to an MLP during training, showing an ability to develop suitable structure whilst learning a task. The use of evolutionary computing techniques to allow for the emergence of appropriate complexity in neural networks has been examined by Harvey et al. [e.g., 1994]. Here an evolutionary gradualism mechanism is used such that the length of the genotypes can increase to an appropriate size over time; extra nodes can be added to the network through a mutation-like operator during reproduction. Stanley and Miikkulainen [2002] have recently suggested a similar scheme. Other population-level techniques which allow for an appropriate increase in genotype complexity include duplication [Lindgren & Nordhal, 1995], where a given genotype has the potential to double in length, and symbiogenesis [Bull & Fogarty, 1996], where genotypes from separate coevolving populations can merge.

The basic concept of neural constructivism can be used within NCS to allow for the emergence of appropriate rule complexity to a given task [Bull, 2002]. Here each rule can have from one up to a maximal fixed number of nodes in its hidden layer fully connected to the input and output layers. At each reproduction event, after mutation, with some probability ($\psi$), a constructivism event can occur in the given offspring. With some probability ($\varpi$) this causes connections to a new node to be added with random weights, otherwise the last connected node is disconnected. The two probabilities are self-adapted in the same way as the mutation rate. That is, each rule has its own constructivism rate and node adding rate which are passed to its offspring. The offspring then apply these rates to themselves (e.g., $\psi' = \psi \times e^{N(0,1)}$), before testing them at the resulting rate. Hence the number of hidden nodes exploited by a given rule evolves over time, at a rate determined by the rule and guided by its interactions with the environment.
5. Results in the Maze Tasks

5.1 Woods 1

Figure 4(a) shows the performance of NCS in Woods 1 where rules started with one connected node in the hidden layer and they can have a maximum of four. The parameters used were as in Section 3.1 and the new parameters were seeded in the same way as the mutation rate. It can be seen that optimal performance is obtained in the simple maze. Figure 4(b) shows the average number of connected nodes in the hidden layer of the rules. Here the neural constructivism mechanism causes an increase in the number of hidden nodes connected before achieving optimal performance, although there is an expected cost in terms of time taken whilst average connectivity increases. On average, rules use two or three hidden layer nodes here. Examination of the resulting rules shows a degree of spatial heterogeneity. For example, in the case noted above where three of the four locations in the top left corner of Woods 1 (Figure 1) are handled by a rule proposing a move southeast, inspection shows that such rules typically contain three hidden nodes. In contrast, the rules for the remaining location, giving a move east, typically contain two hidden nodes. That is, the building of a representation of the problem space has been tailored to the problem by NCS's interaction with it under the constructivism process. Bull [2002] also notes results of this kind using a different constructivism mechanism at a fixed rate.

![Figure 4: a) Performance with neural constructivism in Woods 1](image-url)
Figure 4: b) Average number of nodes connected per rule.

Figure 4: c) Movement of self-adaptive parameters.

Figure 4(c) shows the behaviour of the self-adapting parameters. It can be seen that the mutation rate behaves as expected, declining over the run. The probability of constructivism also displays the same behaviour, reducing at a slower rate. Finally, the probability that a constructivism event is one under which a new node is added levels off at around 0.4, having dropped significantly initially, presumably due to the higher frequency of constructivism events; this parameter becomes selectively neutral as the rate of constructivism diminishes.
5.2 Non-Stationary Woods 1

Figure 5: a) Performance with constructivism within non-stationary Woods1.

Figure 5: b) Average number of connected hidden layer nodes.

Figure 5(a) shows how this form of NCS responds to changes in the learning task (same parameters as before) as it did without the adaptive constructivism. There is a marked increase in the average mutation rate and constructivism parameters (Figure 5(c)) and the average number of hidden nodes after the change (Figure 5(b)). It would therefore appear that mutation again drives re-adaptation, aided by an influx of more complex, new rules.
6. Moving Towards the Robotic Domain

So far this paper has explored the use of NCS within simple grid world environments. The aim of this work is to transfer insights gained from within grid worlds to robotic environments but there are clearly many differences between the two: the grid world is precisely described and delineated while the robotic world is filled with ambiguities. Learning in robotic experimentation must also take place on a considerably different time scale than that typically used in simulation experiments. That is, similar robotic experiments take at best several seconds and at worst several minutes to move from the start state to the reward state and thus runs in the tens of thousands as presented for the grid world domain are clearly impractical. There are two main reasons for having such long runs with the grid-world domains: it demonstrates the stability of NCS performance; and, LCS experiments within maze environments have always used this scale and so it allows comparison to other work.

Figure 6(a) illustrates the performance of NCS without constructivism within the first 1000 visits to the reward in Woods1. It compares performance of NCS and its multi-layered perceptron representation with a radial basis function (RBF) [e.g., Poggio & Girosi, 1990] representation. The radial basis representation is similar to that of the MLP except that instead of encoding a series of weights, a rule encodes a series of centres and spreads (both seeded in the range [0.0,1.0]). It has been previously noted [Bull & O’Hara, 2002] that the RBF representation can give considerably faster learning than the MLP representation which is confirmed in Figure 6(a). Figure 6(b) illustrates how the average matchset size of the RBF representation moves very quickly to an apparently
appropriate equilibrium value of around 100, in contrast to the MLP which moves more slowly from initially much larger matchsets (due to the difference in how the two network types form generalizations). It is of interest that both representations move to the same average matchset size, suggesting that the fitness sharing methodology of ZCS (NCS) is robust.

These results indicate the potential difficulties in using multi-layered perceptrons for NCS robotic research. That is, the MLP representation seems to move from very general rules to more specific ones which, whilst shown above to be effective, is much slower than the RBF’s movement from specific to appropriately general rules. Further, the ease with which the RBF representation can be configured by adjusting the seeding spreads of the rules gives the ability to easily initialise a population to rules of varying specificity over a defined range. Hence, for the real robot experiments presented here, a switch to the RBF representation is made.
7. Robotic Environment

The tasks examined in this paper are three relatively simple robot learning experiments: the first is to learn phototaxis; the second is phototaxis with obstacle avoidance; and the last is non-stationary since the robot must learn phototaxis and then the same phototaxis with sensor disruption. A large amount of effort has been taken to automate the setup so experiments can be run without needing human monitoring or direct involvement (after [Hurst et al., 2002]). Motivation for these particular experiments comes from the only other significant body of work using LCS for the control of real robots, that by Dorigo et al. (see [Dorigo & Colombetti, 1997] for an overview). They used a hierarchical architecture to learn various phototaxis and obstacle avoidance behaviours. However, their work used step-by-step reinforcement – “behavioural shaping” – rather than the standard delayed reward scenario used here.

The robotic platform used in the experiments described is the University of the West of England's "LinuxBot". This is a three-wheeled robot, two of which are powered. It is controlled via a radio LAN system and acts as a probe with motor commands being sent by the LCS algorithm running on a base station. Sensory information takes the form of three real-valued numbers from three light sensors placed on the top of the robot. All sensor values are scaled between 0 and 5. The robot also has three IR proximity sensors placed on the front of the robot (Figure 7(a)). The IR sensors have an effective range from 3cm to 3m, while the light sensors have a range of around 5m. The robot has three actions available, move continuously left, right or forward, where the turning actions are obtained by slowing one of the wheels to half its usual speed.

Figure 7: The LinuxBot platform and the experimental setup.
Figure 7(b) shows the experimental setup where the robot runs on a metallic powered floor, allowing experiments to be run for extended time periods. The lights are surrounded with metal bumpers and only one is on at any given time. When the light is on and the bumper surrounding it is hit, this light is then switched off and the light at the opposite end of the arena is switched on. This switching is monitored by a computer base station, which sends a message to the LCS informing it that the robot has reached its goal state. The robot then automatically reverses and rotates by 180 degrees. The curvature of the light bumpers and the different angles by which the robot can hit the bumpers ensures that the robot does not always have the same initial orientation for the next trial. Figure 7(b) illustrates the experimental area when it is configured for the phototaxis and obstacle avoidance experiments. The arena is a rectangle 2.2 m x 1.7 m and, for one experiment used here, in the centre of the arena is placed a box (0.4m x 0.5m) heavy enough to prevent it from being disturbed by the robot. The optimal time to reach the goal state from a start state is ~15 seconds for the obstacle avoidance task and ~12 seconds for the normal phototaxis task. For the experiments which only consider phototaxis, the box in the centre of the area is simply removed and the input from the IR sensors is disregarded. As well as possessing light and IR sensors, the robot also has two “bump sensors”. When these are hit the robot automatically stops whatever action it is doing and reverses 10 cm and sends a signal to the controlling algorithm.

8. T-NCS: A Neural Learning Classifier System for Robotic Environments

Reinforcement learning methods (see [Sutton & Barto, 1998] for an introduction) typically assign a value to each possible state-action combination of a given task. When a programmer is prepared or able to define the state space \textit{a priori}, this methodology has been proven to work for robot systems [e.g., Asada et al., 1996]. The approach is however labour intensive and becomes less tractable when the state space increases in complexity. There are broadly two accepted methods by which generalization of state spaces can be achieved - gradient descent methods and tiling methodologies [Lin, 1992][Tham, 1995]. These methods only deal with discrete actions. Santamaria et al. [1998] extend this approach by considering continuous duration action spaces, although their approach can not generate continuous value actions. Millan et al. [2002] have an interesting methodology (incremental topology preserving map, IPTM) for generating new continuous actions, but they require the presence of appropriate \textit{a priori} defined “reflexes”, where the continuous space is generated around these predefined reflexes. TCS [Hurst et al., 2002], and its neural derivative T-NCS used here, can develop a
discretisation of the state space, whilst learning actions that are continuous in duration, without recourse to predefined reflexes.

Previously, TCS was shown able to learn simple robotic tasks using a rule representation consisting of floating point numbers to delineate the range of inputs recognised by the rule. This works extends that by considering the use of neural representation and constructivism. TCS has been devised by adding two significant changes to the effective and simple ZCS algorithm - a change in the action selection mechanism and a change in the reward policy. Use of the scheme within NCS is now described.

8.1 Action Selection and Continuation Policy

On receipt of sensory input all rules in the rule-base process the input and those that match the input form a matchset [M] as before. There are three possible actions and hence each rule has three output nodes here; move continuously left, right or forward. An action is selected from the rules contained in [M] and all rules that match the current input and advocate the chosen action are placed in an actionset [A]. Since RBFs are being used, the matching process can be simplified. Rather than include an extra output node, a rule is simply said to match if it gives a positive response on any output node – one per action as before.

T-NCS exploits actions of continuous duration and thus while the robot is carrying out the ordained action, the input from the environment is continually sampled. To decide if T-NCS should continue with an action or drop it, the current input is passed through the rules in the current [A]. In the case where none of the current members of [A] match the input, the procedure is straight forward: the current members of the action set are moved to the previous action set where they receive an internal reward. A new action is then selected. When all the rules in [A] match the current input, the effect is to continue with the current action. The situation where a proportion of the rules match the current input and a proportion do not is more complex. A decision has to be made either to continue with the current action, or stop and consider a different action. The actionset [A] can therefore be divided into two subsets: those rules advocating continuing with the action, the "continue set" [C]; and those advocating stopping the current action, the "drop set" [D]. Simple roulette wheel selection is then carried out over the entire actionset based on fitness as before. If the rule selected is from the [D], all members of [C] are removed from [A] and the contents of the actionset are moved to the previous actionset, where they receive
internal reinforcement. If however the rule is selected from [C], all rules contained in [D] are removed from the current actionset. The robot continues with the action until a drop decision is taken or an external event occurs. A further test must be made to see if the action proposed by rules in [C] has changed. That is, under the neural representation rules can change which output node has the highest activation with only a slight change in input. Therefore, the action with the highest total fitness in [C] is chosen and the remaining rules are then treated in the same way as the dropset. The external events that can break an action-continue cycle are:

1. The robot hitting an obstacle.
2. The robot hitting the light switches.
3. The robot continuing with an action longer then a maximum predefined limit.

8.2 Reinforcement Learning Policy

Reinforcement learning within T-NCS is essentially as before, i.e.:

\[
S_{[A]}' \leftarrow \beta r_{\text{sum}} + \gamma S_{[A][t]}
\]

The change to this update procedure is that the external and internal reward received by the system is not discounted by a fixed factor but by a variable factor depending upon the time taken by the system to reach the current state. In the case of the external reward the total time taken from the start state to the goal state is used to discount the reward. With the internal reward the time taken is the time to move from one state to the next. In this way the reinforcement learning process moves away from generalizing over Markov decision state spaces to generalizing over Semi-Markov Decision Processes (see [Parr, 1998]). More formally:

\[
S_{[A]}' \leftarrow e^{-\phi t} r_{\text{sum}} + e^{-\eta t'} S_{[A][t]}
\]

where \( t' \) is the total time to achieve the task, \( t \) is the duration of the action, \( \phi \) and \( \eta \) are their respective discount factors. The overall effect of this algorithm is for the LCS to learn appropriate discretizations in the continuous input space, as it solves the given reinforcement learning task, exploiting its population of rules developed under the GA.
As in ZCS, taxation is also carried out on all members of the [M] who do not go on to form the [A]. TCS extends this tax to those classifiers which form the [D], if a decision has been made to continue with the action. This taxation takes the form of removing a fraction ($\tau$) of fitness from each unselected member of the matchset or the contents of the dropset. Finally, the GA is (always) fired at the end of a trial and not during a trial as this was found to cause some disruption. The GA is fired twice, i.e., four offspring are created per trial.

8.3 General Parameterization

The experiments were carried out with different initialisations of the construction parameter settings. In the first two experiments reported here, the self-adaptive parameters were initialised around known good values. These values were obtained from the previous work in the grid world environments, i.e., the construction rate was initialised around 0.01 and the node addition rate was initialised around 0.5, with the number of hidden layer nodes initialised to 3. In the second pair of experiments the construction parameters were seeded randomly in the range [0.0,1.0] as in the simulations, with the adaptive mutation rate set (conservatively) in the range [0.0-0.01]. Wider settings of the adaptive mutation rate were not used throughout as it was found that this prevented learning in a reasonable time period; typically, learning only occurred after 600 visits to the goal state when the adaptive mutation rate was seeded [0.0-1.0]. The parameters controlling T-NCS are detailed in Table 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>1500</td>
<td>Number of rules</td>
</tr>
<tr>
<td>$\beta$</td>
<td>0.85</td>
<td>Used to form bucket</td>
</tr>
<tr>
<td>$\tau$</td>
<td>0.05</td>
<td>Taxation rate used to penalize unselected rules</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.5</td>
<td>Discount applied to reward</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>0.05</td>
<td>Discount applied to Bucket</td>
</tr>
<tr>
<td>$f_{\text{max}}$</td>
<td>10000</td>
<td>External reward</td>
</tr>
<tr>
<td>$\mu_0$</td>
<td>0.005</td>
<td>Average initial seeding of mutation rate</td>
</tr>
<tr>
<td>$f_i$</td>
<td>10</td>
<td>Initial fitness</td>
</tr>
<tr>
<td>$\psi_0$</td>
<td>0.01</td>
<td>Average initial Construction rate</td>
</tr>
<tr>
<td>$\omega_0$</td>
<td>0.5</td>
<td>Average initial neuron addition rate</td>
</tr>
<tr>
<td>$\nu_0$</td>
<td>0.5</td>
<td>Average initial setting of RBF centres</td>
</tr>
<tr>
<td>$\sigma_0$</td>
<td>0.0175</td>
<td>Average initial RBF spread parameter</td>
</tr>
<tr>
<td>$w_0$</td>
<td>0.0</td>
<td>Average weight value set in range [-0.5,0.5]</td>
</tr>
</tbody>
</table>
9. Results

All results presented are the average of five runs, unlike the simulations where ten runs were used. The robotic results also do not display a running average, which further contributes to their “spiky” nature but clearly indicates the robot’s online performance.

9.1 Phototaxis

Figure 8(a) shows the performance of T-NCS on the simple phototaxis task, with all self-adaptive parameters seeded at values from the simulations as discussed above. It can be seen that the time taken to reach the goal drops to near optimality after about 150 trials but there is continued variation. This can be attributed to several reasons: the action selection policy is not particularly greedy (e.g., as shown in the simulations above) but this allows constant online adaptation; the GA is constantly introducing new rules, again to maintain the learning ability of the system; and, the goal state has only been visited a relatively few number of times and so unexplored aspects of the environment are being constantly experienced. Figure 8(b) shows how the number of hidden layer nodes used by the rules remains constant and, correspondingly, Figure 8(c) shows how the self-adaptive parameters remain constant. This last result is not unexpected since they were seeded at apparently good values.

![Figure 8: a) Performance of T-NCS on the phototaxis task.](image-url)
9.2 Phototaxis with Obstacle Avoidance

Figure 9 shows the performance of T-NCS on the phototaxis task with an obstacle placed in the middle of the arena (Figure 7(b)). Here rules contained five hidden layer nodes and all self-adaptive parameters were seeded as before. Note that there are six input nodes for this task, as opposed to three in the normal phototaxis task.
It can be seen that a similar level of near optimal online performance is obtained. Again, the adaptive mutation rate and constructivism parameters do not move significantly from their initial seeding values (not shown). This experiment demonstrates that the system can learn a relatively complex task. However, the time taken for these experiments was considerably longer and hence only the phototaxis task is used for the following experiments which are more closely based on those in the simulations.

9.3 Phototaxis with Random Seeding of the Constructivism

In Section 9.1 the constructivism parameters were seeded around values obtained from the simulations and the number of hidden nodes within the rules was set at an appropriate amount. Figure 10(a) shows the performance of T-NCS when the self-adaptive constructivism parameters are seeded uniformly randomly in the range [0.0,1.0] and rules are initialised with one hidden layer node, as in Section 5. The performance is actually slightly better than that shown in Section 9.1 (Figure 8(a)). The run is continued over a longer period of time (700 visits to the goal state in total) to show the extent of parameter self-adaptation. Figure 10(b) shows how the average number of nodes in the hidden layer rapidly increases during the first 350 trials and then begins to plateau. The movement of the constructivism parameters (Figure 10(c)) follows the previous trends of the simulation work by decreasing in value as the run progresses. The run is not long enough to demonstrate any great movement but it is of a similar extent to that in the simulations over the same time period. It should be noted that although the population was initialised with only one hidden layer node the performance of the system is not significantly worse than that of the system initialised with three hidden nodes. This suggests that the problem can be solved by a range of network architectures and that use of a constructivist scheme allows a
minimal solution to be produced. Again, the mutation rate seeded at the appropriate value does not change to any marked degree (not shown).

![Figure 10: a) T-NCS performance with random seeding.](image)

![Figure 10: b) Average number of hidden layer nodes.](image)

### 9.4 A Non-Stationary Task: Phototaxis with Sensory Disruption

To examine the behaviour of the system on the mobile robot further, in line with points raised in the introduction and Sections 3 and 5, a non-stationary version of the phototaxis task was used. The parameters were set as in the previous experiment (i.e., starting with a single node and construction parameters initialised in the range [0.0-1.0]). Here the robot is given 350 trials to learn the phototaxis task, after which the robot’s light sensor inputs are swapped. That is, the input to the left sensor is presented to the node which was originally taking input from the right sensor and vice-versa, and the robot is then given another 350 trials to re-adapt.
Figure 11(a) shows how T-NCS had learnt the original phototaxis task to the same extent as above before experiencing a significant drop in performance at the point of change, which the system recovers from at around the 600th visit to the goal state. Figure 11(b) indicates the mean number of hidden layer nodes of the rules which differs from Figure 10(b). In the latter case, the graph shows a levelling out at around 1.8 nodes. In the former case, it can be seen that at the end of the experiment the population’s mean hidden layer node number is around 1.94 and still displays a marked trend of increasing. This is akin to the results from the grid world simulations where an increase in hidden nodes was seen after the change (Figure 5b). Figure 11(c) demonstrates the movement of the constructivism parameters, where there is a slight difference in the movement of the addition rate which appears to maintain its value (compare Figure 11(c) to Figure 10(c)); with a change, the rate of node addition drops more slowly than without a change, indicating again, that the system’s response to the change in the environment is to increase rule complexity. The adaptive mutation rate, in contrast to the experimentation within simulation, shows little change despite the dynamics in the sensory environment. A possible explanation for this is that in the simulations the change is made after 15,000 trials and that, by running the experiment for so long before the point of change, the population has almost converged to a solution. Hence, the population needs the increased mutation rate to re-introduce diversity into the population. In contrast, in the case in the robotic experiments there is perhaps still a significant amount of diversity in the rule-base as the GA has not operated so many times.
Analysis of the resulting rules does not show any clear relationship between the number of hidden layer nodes contained within a rule and where that rule is used, as was seen in the simulations. However, this may simply be due to the much shorter timescales involved.
10. Conclusions

The Neural Learning Classifier System approach to artificial learning entities would appear to, potentially at least, encompass many of the key aspects raised in the introduction. This paper has explored the use of self-adaptive constructivism within the architecture as an approach to aid the realization of complex/appropriate autonomous behaviour, exploiting NCS’s basis in evolutionary computing.

The results presented here indicate reasonable performance for NCS in simple robotic tasks and adds to the previous work demonstrating that learning classifier systems have the ability to work in continuous space and time within robotic environments [Hurst et al., 2002].

We are currently extending this work with the robot to incorporate gradient descent methods for determining the parameters of the RBF rules to increase the rate of learning (after [O’Hara & Bull, 2003]). Further, since NCS uses neural networks to generalize at the level of state-action pairs it can be used with continuous value actions without alteration (see [Bull, 2002]) and this is also under consideration. Other methods for constructivism are also being explored (after [Redding et al., 1997]), along with an accuracy-based fitness scheme for the NCS [Bull & O’Hara 2002]. Additional problems also exist with regard to the length of time taken for the algorithm to converge. This is a noted problem for reinforcement learning methodologies and eligibility traces (such as Q(\(\lambda\)) [Peng & Williams, 1994]) are a recognised method to speed convergence. The incorporation of eligibility traces into the T-NCS framework may produce a similar beneficial speedup.

Acknowledgments

We would like to thank Ian Gillespie, Ian Horsfield and Chris Blythway of the Intelligent Autonomous Systems Laboratory, UWE Bristol for their technical assistance. Thanks also to the members of the Learning Classifier Systems Group at UWE for many useful discussions. This work was supported under EPSRC ROPA grant no. GR/R80469.
References


