XCS with Combined Reward Method (XCSCR) for Policy Search in Multistep Problems

Zheming Zhang
Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
tonyzhang@ecs.vuw.ac.nz

Will N. Browne
Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
will.browne@vuw.ac.nz

Dale A. Carnegie
Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
dale.carnegie@vuw.ac.nz

***Abstract*—A reward mechanism is critical for a Reinforcement Learning agent to learn action policies from rewards. The reward mechanism establishes a policy by estimating contributions of constituents of the policy to a reward. Traditionally, rewards from an environment have two categories: long-term rewards for guiding the policy learning process, and short-term rewards for optimisation. However, long-term, positive rewards are scarce at the initial learning phase in multistep problems such that existing reward mechanisms lack sufficient stimulus to learn policies effectively. This paper proposes *XCSCR*, an Accuracy-based Learning Classifier System (XCS) algorithm with a combined reward (CR) method, to guide the search for global optimal policies in multistep maze problems. The XCSCR discriminates long-term and short-term rewards through four novel reward-assignment mechanisms: 1) A short-term reward mechanism encourages exploration of the RL agent searching for policies based on short-term rewards. 2) An imprinting mechanism amends the negative impact of indiscriminate rewards between exploration and exploitation. 3) A learning-rate switching mechanism emphasises the impact of long-term positive rewards in the policy searching process. 4) A learning step-threshold mechanism creates an optimisation pressure for policies. Experiments were conducted in three maze environments as this enabled the effects of XCSCR on policies to interpreted easily. Results show that the XCSCR enables learning the optimum path-finding policies quicker and more often than previous XCS algorithms. The XCSCR's improvements for the policy search will facilitate real-world applications, e.g. robotic applications.**

***Index Terms*—reward mechanism, scarce/sparse reward, Reinforcement Learning, multistep, maze problems, XCS, XCSCR**

I. Introduction

A Reinforcement Learning (RL) agent seeks to learn to improve performance from its interactions with an environment. The RL agent performs actions in an environment and receives rewards as feedback from the environment. These rewards are assigned by a reward method within the agent to estimate the contributions of each action. Actions that would contribute to the maximum reward are selected as optimal policies as estimations become accurate. By successively applying learned policies, an agent can optimise its performances in the environment.

An RL agent often faces a vast policy space when searching for a global optimal policy in a multistep environment. A multistep environment, such as a way-finding maze, requires an RL agent to take steps and perform sequential actions to complete a task. For example, the agent is required to complete a navigation task by choosing four directions within ten steps in the maze 4 environment (see Section IV-A). There are three global optimal policies compared with a policy space with naïvely 4^{10} policies (the number of possible combinations of four actions that the agent can attempt to take in ten steps). The vastness of a policy-searching space increases the difficulty to solve the *credit assignment* problem. The credit assignment problem was introduced by Minsky [1]: "If a sequence of rules fires before the system solves a particular problem, how can credit or blame be accurately assigned to early rules that set the stage for the final result? [2]" Specifically, the vastness of a policy-searching space requires efficient estimations of the worth of actions within policies to search for an optimum policy.

The standard reward methods can not efficiently estimate actions and policies when positive rewards are scarce [3]. An estimation relies on assigned rewards through the Bellman's equation (see Section II-C), which does not discriminate long-term rewards between positive ones and negative/neutral ones. Actions and policies can be estimated effectively when positive rewards are frequently achieved as they guide the search. If the number of the negative/neutral rewards is overwhelming, the estimations generated by scarce positive rewards will be quickly offset by the interference of the negative/neutral rewards. As the reward method fails to estimate actions and policies efficiently, global optimal policies are elusive to the RL agent in multistep problems.

These problems are particularly apparent in real-world robotics experiments where it is impractical to conduct millions of iterations, which are common in simulations. Similarly, although modern computing power is increasing, those available to an autonomous robot is often limited. Previous work by Williams et al. [4] [5] has adapted Learning Classifier Systems for policy learning in real-robots (see Section II-B), but this work still suffered from credit assignment problems in relation to scarce rewards in a real-world environment. In addition, the work did not take into account the benefits of an autonomous agent in terms of being able to identify short-term reinforcement, e.g. the ability to note a collision during a policy learning trial. Therefore, a reward method that emphasizes these credit assignment problems would be

978-1-7281-2153-6/19/$31.00

beneficial to real-world robotic applications.

This paper proposes an XCSCR, a combined reward (CR) method for an Accuracy-based Learning Classifier System (XCS) algorithm, to achieve global optimal policies in multi-step problems. That is, XCSCR needs to introduce immediate short-term negative rewards to encourage exploration when long-term, positive rewards are scarce. XCSCR also considers to exploit the policies that led to long-term, positive rewards to balance potential interferences from long-term, negative rewards. In addition, XCSCR emphasises the effect of long-term, positive rewards, and includes a dynamic threshold to drive policies to move toward globally optimal ones. The effects of XCSCR on policy search will be demonstrated with experiments using interpretable multistep maze problems.

II. Background

A. XCS

Learning Classifier Systems (LCSs) are rules-based machine learning algorithms. An LCS agent learns interpretable "condition-action" rules, termed classifiers, through interactions between the learning agent and its environment. These interactions allow the LCS agents to estimate the worth of rules through various criteria. An Accuracy-based Learning Classifier System (*XCS*) is an LCS algorithm that estimates the worth of rules based on the accuracy of the prediction of these rewards. An XCS agent can learn to complete a task as these estimations become accurate.

B. Standard XCS Learning Iteration

A standard XCS agent estimates rules through iterations of these interactions. Based on how the agent perceives the environment in each iteration, rules in the XCS agent will advocate an action as the agent's effect on the environment (see Figure 1.a). All rules in the XCS rules' population [*P*] that match the current perception will form a Match Set [*M*] through the match filter. Next, [*M*] selects an action by the selection filter through either an exploration or an exploitation method. In exploration, the selection filter selects an action randomly from all options. In exploitation, the selection filter selects the most promising action with the maximum worth in [*M*] (see Equation 6). The selected action will be executed by the agent, and rules in [*M*] who vote for the executed action will form an Action Set [*A*]. After the action execution, the agent will immediately receive a step reward r_a from the reward filter as feedback from the environment in single step problems. The worth (fitness) and other statistics of the rules in [*A*] are updated according to the step reward r_a. As iterations progress, fitness better reflects the utility of a rule to guide evolutionary search to better solutions (policies of actions).

The reward filter updates each rule in [*A*] by updating statistics of a rule, including predicted reward r_p, prediction error ϵ, and fitness F based on its r_a. Predicted reward r_p is updated by a learning rate β (see Equation 1).

$$r_p = r_p + \beta * (r_p - r_a) \tag{1}$$

Prediction error ϵ is also updated in a similar way (see Equation 2).

$$\epsilon = \epsilon + \beta * (|r_p - r_a| - \epsilon) \tag{2}$$

Finally, fitness fit, the worth of the rule, is updated through calculations of absolute accuracy κ and relative accuracy κ' (see Equations 3, 4, 5).

$$\kappa = \begin{cases} 1, & if\ \epsilon \leq \epsilon_0 \\ (\epsilon/\epsilon_0)^{\nu}, & otherwise. \end{cases} \tag{3}$$

$$\kappa' = \kappa / (\textstyle\sum_{[A]} \kappa) \tag{4}$$

$$fit = fit + \beta * (fit - \kappa') \tag{5}$$

$$worth_{a_j} = \frac{\sum_{cl_k \in [M] | a_j} r_{p_k} * fit_k}{\sum_{cl_k \in [M]} fit_k}, \tag{6}$$

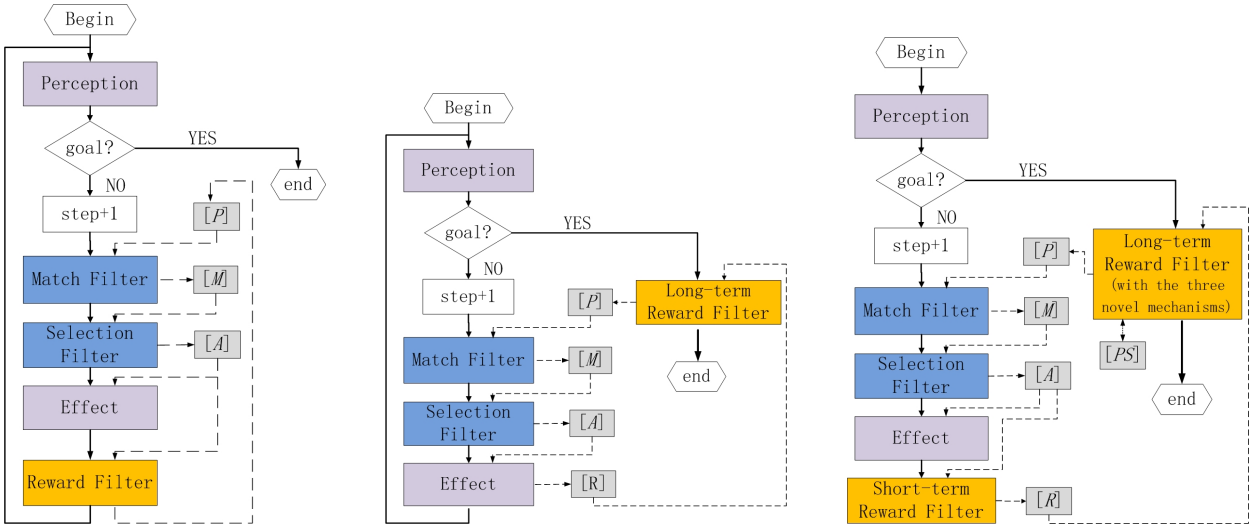
where j,k $\in \mathbb{N}$, *cl* is a classifier, a_j is the action of *cl*, fit is the fitness of *cl*, r_p is the predicted reward of *cl*, $worth_{a_j}$ is applied to a rule set [*M*].

C. Multistep problem and Robotics Adaption

XCS algorithms have been applied to solve multistep problems [6], such as maze problems introduced by Wilson [7]. In maze problems, the agent has to execute actions in sequential iterations to complete a navigation task. Rewards are provided only in certain states in the multistep problems. A long-term reward r_l represents the completion of the task. r_l equals an arbitrary value of 1000 if the task is completed. Otherwise, r_l equals zero in non-goal states or if the task fails (e.g. the agent has taken more steps than the step-threshold in an epoch of a trial). A short-term reward (immediate reward) r_s traditionally represents the agent's execution effects during each iteration. r_s equals an arbitrary value of -1 as a cost of the agent moving a step forward. r_s equals an arbitrary value of -50 if a collision occurs.

Traditionally, a reward propagation is required by a reward method for applying the standard XCS iteration loop in the multistep problems. The reward method is internally responsible for assigning external rewards, including short-term rewards and any long-term reward, to step rewards for each iteration. The step reward r_a is the reward which the agent assigns to recent active rules in $[A]_i$ for the step i, as mentioned in the single step problem above. The standard XCS reward method encapsulates long-term rewards into short-term rewards through a Q-learning like algorithm [6] [7]. The maximum potential reward available is propagated to previous actions $[A]_{i-1}$, even if that action is not taken in $[A]_i$. Although Q-learning can mathematically guarantee the approximation of estimated r_a, it might take a long time before the effect of a r_l can propagate to r_as of early states. The propagation speed of the standard reward method might be not fast enough for robotic applications in the real-world.

Williams et al. [4] [5] proposed a reward method suited to robotic applications to solve multistep problems. The reward method encapsulates short-term effects, such as collisions and



(a) Standard XCS Iteration Loop (b) Robotic XCS Iteration Loop (c) XCSCR Iteration Loop

Fig. 1: XCS algorithms' learning iteration

Blue blocks are the standard XCS sub-processes, grey blocks are XCS rule sets, orange blocks are sub-processes of reward methods, and purple blocks indicate the agent's interactions with a maze environment. Arrows indicate the agent's working flow, and dotted arrows indicate classifiers/rules working flow.

number of steps taken, into the time factor (*time*) of long-term rewards (see Equation 7). Instead of updating [*A*] at the end of each iteration as the standard XCS reward method does, the robotics method records [*A*] in a Reward Stack [*R*] (see Figure 1.b). At the end of a task, the robotic reward method propagates the long-term reward r_l backward to all previously active XCS rules evenly (see Equation 8) or with a discount factor (γ) to emphasise the contributions of recently active rules (see Equation 9).

$$r_l = 1000 + 1/time \tag{7}$$

$$r_{a,i} = c * r_l / num_{step}, \quad \text{where c is a constant, and i} \in \text{step} \tag{8}$$

$$r_{a,i} = r_l * \gamma^i, \quad \text{where i} \in \text{step} \tag{9}$$

D. Related Work

Nakata et al. [11] proposed XCS with Adaptive Action Mapping (*XCSAM*) to solve multistep problems. XCSAM targets these problems by focusing evolution on the best actions. XCSAM sorts actions by their worth (see Equation 6), and selects classifiers/rules which contain the actions with the best worth, as best action mappings. XCSAM thus focuses on the evolution of the best action mappings to search for action policies. XCSAM records the effects of scarce positive rewards on the estimation of the worth of actions/rules. Instead of improving the accuracy of the estimation, XCSAM applies the estimation directly for the evolution and relies on the evolution to guide search in the vast policy space.

XCSAM may apply transferable policies learned in a smaller policy space to learn a bigger policy space. Although experiments were conducted in the same maze 6 environment, the starting cell was randomly placed. The closer to the targeted cell a starting cell is, the smaller the policy space to search. Policies that are learnt in a smaller policy space can be transferred and applied to the policy search in a larger space. To rule out the effects of the transferable policies, this work will require a fixed end-to-end experimental setting.

Deep Reinforcement Learning (deep RL) also faces challenges of sparse rewards. Behavioural cloning is applied in deep RL for policy search when the rewards are sparse or a reward function is unavailable [3] [8]. Behavioural cloning learns policy from mapping state-action pairs from expert trajectories, without learning the reward function. Recently, in robotics applications, the sparse final reward problem was addressed by augmenting auxiliary rewards. Mirowski et al. [9] added auxiliary rewards to facilitate an end-to-end maze navigation task. The auxiliary reward includes randomly placed fruits to encourage exploration. Similarly, this work will include negative rewards, such as punishments for collisions, as auxiliary rewards to facilitate a policy learning process. Deep RL often requires a large computational budget for both learning and interpreting the policies, which restricts its use on small autonomous robots.

III. Method

This paper proposes XCSCR, an extension of XCS with a combined reward (CR) method, to guide the search for global optimal policies in multistep problems. CR discerns the impact of various long-term and short-term rewards to

address different challenges arising from the scarcity of long-term, positive rewards. Its combined reward method includes a short-term reward mechanism and a modified long-term reward mechanism. The short-term reward mechanism aims to encourage exploration at the learning agent's initial training phase, when the agent faces a vast policy searching space. The long-term reward mechanism aims to exploit the long-term, positive reward effectively. The long-term reward mechanism is developed from the standard reward method with three novel reward mechanisms: an imprinting mechanism, a learning-rate switching mechanism, and a learning step-threshold mechanism. The imprinting mechanism aims to balance the negative effects of an agent's explorations on current best policies in multistep problems. The learning-rate switching mechanism differentiates long-term rewards between positive ones and negative ones. The learning step-threshold mechanism aims to create an optimisation pressure to drive local optimal policies toward global optimal policies.

A. Short-term Reward Mechanism

Short-term rewards are defined as immediate rewards that the agent receives in each step. Traditionally, in a Markov model, short-term rewards are arbitrary negative values that demonstrate immediate effects of an agent's recent step [10]. If the agent runs to an obstacle-free cell, a small negative value, such as -1, will be assigned to this step as its cost. If a step leads to a collision, another negative value with a large magnitude, such as -50, may be the short-term reward as a severe punishment to the step. By taking short-term rewards into an optimisation function, a training process would drive local optimal policies toward global optimal policies.

The reward encapsulation of the adapted robotic reward method [4] [5] fails to identify the rules that directly respond to short-term rewards (see Section II-C). The effects of short-term rewards were encapsulated as a time factor (see Equation 7), which sums up the agent's short-term performances in the standard reward filter (see Figure 1). Although the time factor encapsulates effects of collisions if they happened during a trail, the time factor fails to record when these negative rewards happened. Thus, information to identify responsible rules is neglected by this method.

As the adapted robotic reward method fails to identify the rules responsible for the negative rewards, this leads to a purely random search in an agent's initial training phase. Before the establishment of effective policies, long-term, positive rewards are scarce and the agent frequently receives long-term, negative rewards. All the long-term rewards are indiscriminately propagated to rules as the reward method suggests (see Equations 8 and 9). The overwhelming majority of negative rewards could be unintentionally propagated to rules that could otherwise lead to positive rewards. Therefore, all rules could be estimated as equally bad (see Figure 2). In this case, the agent will essentially take purely random actions because explorations rely on the difference between the worth of rules.

In contrast, this paper proposes the short-term reward mechanism that attributes any severe negative short-term rewards only to the latest activated rules. The short-term reward mechanism is implemented by a short-term reward filter, which is inserted at the end of an XCS iteration loop (see Figure 1.c). The short-term reward filter aims to immediately respond to severe negative short-term rewards, such as severe collisions, at the current step/state (see Algorithm 1). In the filter, the worth of the responsible rules in an [*A*] will decrease by adding a short-term reward r_s, a negative reward in this case. Then, the calculated worth will be applied to update [*A*] through the XCS standard rules' updating procedure (see Equation 6 for calculating an action's worth from a chosen rule set). The updated [*A*] is recorded in a [*R*] for the long-term reward mechanism at the end of the iteration loop. In addition, minor negative short-term rewards, such as a step's cost, could be ignored through setting parameter θ_{r_s} for saving computing budget.

The short-term reward mechanism bootstraps the policy search when long-term positive rewards are scarce in the initial training phase. The mechanism can differentiate potential useful rules from useless ones, because the useless rules are responsible for severe negative effects and their worth decreases. Even if the long-term positive rewards are scarce, this differentiation can encourage the agent to explore the targeted states appropriately (see Section IV and Figure 5.a).

algorithm 1 Short-term Reward Filter

Inputs: Short-term Reward r_s, Action Set $[A]$, Population$[P]$.
Outputs: Action Set $[A]$.

```
1: function UPDATE SHORT-TERM REWARD(r_s, [A])
2:     if r_s <= θ_{r_s} then
3:         a_j ← [A]        //get the last executed action a_j
4:         worth_{a_j} ← (a_j, [P])
5:         worth_{a_j} = worth_{a_j} + r_s
6:         Call Function: reward procedure ([A], worth_{a_j})
7:     end if
8:     return [A]
9: end function
```

B. Imprinting Mechanism

An exploration and an exploitation method are common optimisation components for machine learning algorithms, including XCS algorithms. In XCS algorithms, the exploration and the exploitation are applied to select an action in the selection filter (see Figure 1). In exploitation, the filter will select the action with the best action worth, $worth_{a_j}$ (see Equation 6). The filter will also activate exploration with a probability parameter p_{explr} to choose an action purely randomly. Previous studies show that a sensitive balance between the exploration and the exploitation is critical for XCS to learn optimal policies in single-step problems [12] [13].

However, the exploration and the exploitation can hardly achieve the sensitive balance that drives XCS to learn optimal policies efficiently in multistep problems. This is because

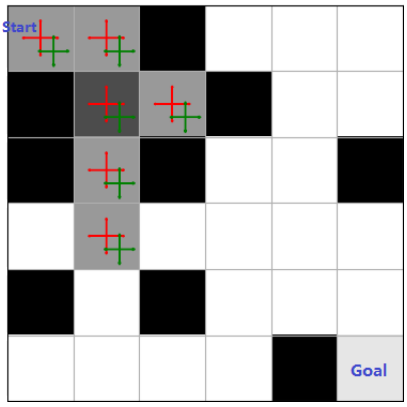


Fig. 2: Worth of Rules (actions). When the long-term, positive rewards are scarce, all actions in initial steps are equally bad. Red arrows: worth of rules, magnitude suggests the negative value of the worth. Green arrows: optimal actions/policies, magnitudes are fixed. Light grey cells suggest they have been visited by the agent in this epoch of a trail. The dark grey cell suggests the location where the agent ended up at the end of the epoch.

algorithm 2 Imprinting Procedure

Inputs: Current Policy p, Policy Set $[PS]$, long-term reward r_l, Population Set $[P]$.
Outputs: Policy Set $[PS]$, Population Set $[P]$.

function IMPRINTING PROCEDURE(p, $[PS]$, r_l)
  if r_l == 1000 **then** // p achieved the goal position
    Call Function: update $[PS]$ procedure (p, $[PS]$)
    return $[PS]$
  else if ($RandomNumber[0,1) > \theta_x$ and $[PS]$ is not empty **then**
    // When p failed to achieve the goal position, a learning from $[PS]$ will be randomly actived.
    randomly select a Policy p_i from $[PS]$
    Call Function: behavioural cloning procedure (p_i, $[P]$)
    return $[P]$
  end if
end function

the standard reward-assignment mechanisms for multistep problems do not differentiate explored steps and exploited steps when the mechanism propagates rewards [6] [10]. The Q-learning reward-assignment mechanism, the even reward-assignment mechanism (see Equation 8) and the discount reward-assignment mechanism (see Equation 9) propagate rewards based on the sequence of steps (see Section II-C). Thus, effects of exploration and exploitation activated in different steps can be interfering with each other. For example, an iteration could fail because of an explored action, which is activated by the probability of p_{explr}, in a step. Without an additional algorithm to differentiate steps between exploration and exploitation, rules, which contribute to exploited actions, would be punished with the explored rules for the failure. Since the exploration will be activated with the probability of p_{explr}, the best rules, which are in the majority of exploration steps, will be affected. Therefore, global optimal policies, which consist of the best rules, are hard to achieve effectively or be stable in a multistep XCS agent.

algorithm 3 Updating $[PS]$ Procedure

Inputs: Current Policy p, Policy Set $[PS]$.
Outputs: Policy Set $[PS]$.

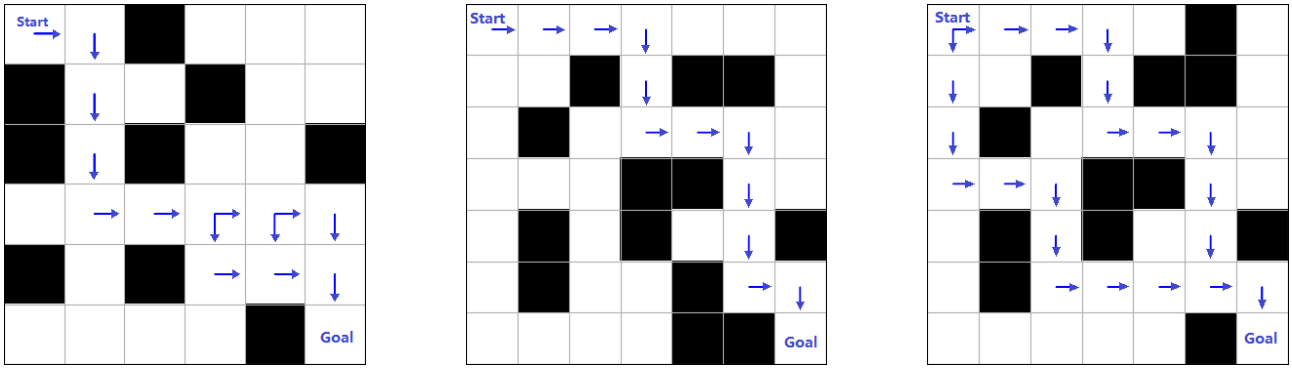
function UPDATING $[PS]$ PROCEDURE(p, $[PS]$)
  if $[PS]$ is empty **then**
    initialise $[PS]$ with p
  else if $p.length == p_i.length$ ($p_i \in [PS]$) and $p \notin [PS]$ **then**
    cover p by $[PS]$
  else if $p.length < p_i.length$ ($p_i \in [PS]$) **then**
    $[PS] \leftarrow empty$
    cover p by $[PS]$
  end if
  return $[PS]$
end function

algorithm 4 Behavioural Cloning Procedure

Inputs: Selected Policy p_i, Population Set $[P]$.
Outputs: Population Set $[P]$.

function BEHAVIOURAL CLONING PROCEDURE(p_i, $[P]$)
  Behavioural Cloning Set $[BC] \leftarrow empty$
  for each *state-action* pair in p **do**
    $[BC] \leftarrow ([P] \cap$ *state-action* pair)
  end for
  Call Function: reward procedure ($[BC]$)
  return $[P]$
end function

An imprinting mechanism seeks to restore exploration and exploitation balance by emphasising exploitation of the current optimal policies. Similar to ideas in the deep RL's behavioural cloning, the imprinting mechanism increases the worth of rules that applied in the current best policies. This will be achieved by inserting an imprinting procedure to the long-term reward filter (see Figure 1.c). The imprinting procedure manages a Policy Set $[PS]$, for exploitation reinforcements (see Algorithm 2). When a policy emerges during iterations, the imprinting procedure will update the $[PS]$ through an updating PS procedure to keep the current best policies (see Algorithm 3). When the agent fails in an iteration because of explorations, the imprinting procedure will active $[PS]$ with a probability θ_x (e.g. θ_x=0.5) to emphasise exploitation. The emphasising process will randomly choose a current best policy from $[PS]$ for behavioural cloning procedures. A behavioural cloning procedure allows rules, which led to the chosen policy in $[P]$, to increase their worth as if they had led to a current successful iteration (see Algorithm 4). Through



(a) Maze 4
three global optimal policies

(b) Maze 5
one global optimal policy

(c) Maze 6
two global optimal policies

Fig. 3: Three Maze Environments
Start position is at the left, top cell of each maze environment, and targeted position is at the right, bottom cell. Global optimal policies/rules are shown with blue arrows.

these procedures, the imprinting procedure allows [*P*] to learn from "expert policies". The worth of rules that have been interfered with by the exploration method can recover their worth in an implicit way. In future, both the updating PS procedure and the behavioural cloning procedure can employ individual XCS agents for their learning and evolving (see Section II-D). XCS agents in these two procedures can co-evolve with an XCSCR agent at different hierarchies.

C. Learning-rate Switching Mechanism

Learning-rate, which specifies the percentage of the updating parameter, can significantly impact an agent's training progress. For example, in equation 1, a learning-rate β specifies the predicted reward's updating process. If β is set to a large value, the predicted reward r_p will rely on the current r_a more than the historic estimation. Otherwise, if β is set to a small value, the predicted reward r_p will have a tiny update toward the current r_a. Traditionally, the learning-rate is well-tuned and pre-set as a fixed parameter for the XCS algorithm.

The learning-rate switching mechanism is proposed to compensate for the scarcity of long-term, positive rewards by switching the learning-rate to appropriate values in the training cycle. To encourage XCS rules learning aggressively from scarce, positive results, the β in equation 1 is set to a large value, such as 0.2, when the agent receives long-term, positive rewards. Negative rewards are more frequent than the positive ones in a training process. When the agent receives a punishment, the β is switched to a small value, e.g. 0.05, to learn from punishments slowly. The reason for this discretion is to avoid an optimal rule in a step of an iteration being overwhelmed by "wrong rules" applied in the same iteration. As a result, the learning-rate switching mechanism emphasises the role of scarce long-term, positive rewards.

D. Learning Step-threshold Mechanism

Step-threshold was introduced into multistep learning agent as a threshold for computing resources. The agent needs to terminate a training iteration before a memory crash if the agent has taken too many steps in a vast space. Step-threshold is a parameter which specifies the maximum number of steps allowed for the agent to search in an iteration. Once the agent takes a number of steps over the value of step-threshold, the agent terminates this trial with a failure to achieve the target. Traditionally, the step-threshold is pre-set according to the property of an environment.

Learning step-threshold mechanism applies this threshold to generate pressure for better policies. The step-threshold θ_{step} is set as a parameter that is learnt from discovered optimal policies. Following the Bellman's function, the step-threshold is updated whenever it is larger than the number of steps $step_a$ taken in the currently applied policies (see Equation 10). When step-threshold approaches the number of steps taken in the global optimal policies, it will create an optimal pressure driving local optimal policies toward global optimal ones.

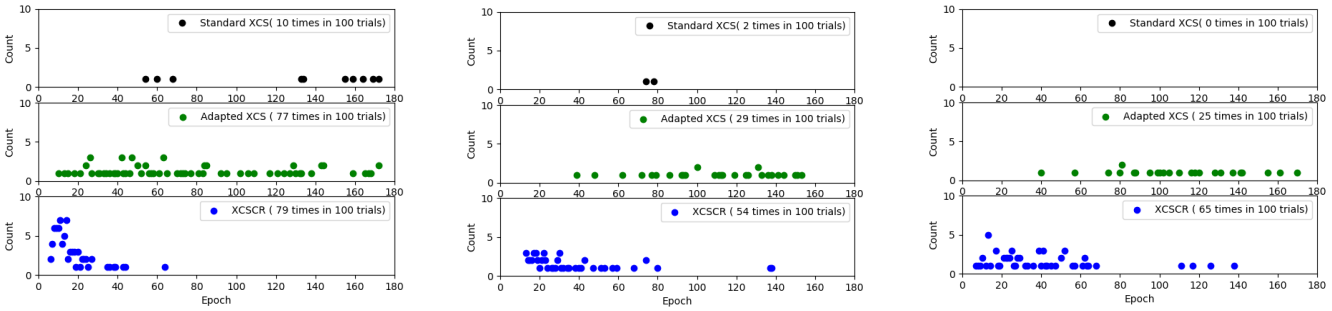
$$\theta_{step} = \theta_{step} + \beta * (step_a - \theta_{step}) \tag{10}$$

IV. Results

A. Experimentation

Maze problems provide suitable environments to test an algorithm's performance on searching for global optimal policies. Maze environments include a large policy space but with known and interpretable optimal paths as policies. This work includes experiments in three maze environments: the maze 4, 5, and 6 (see Figure 3). These maze environments have been frequently applied to demonstrate XCS performance on multistep problems [7] [12] [14]. Maze environments contain free paths, obstacles, a starting position and a targeted position. Free paths are shown as white cells that can be occupied by an agent. Obstacles are black cells.

The navigation task requires an agent to start at the left, top cell of the maze, targeting to the right, bottom cell. In each step (XCS iteration) of an iteration, the agent can choose one direction from four options (left, right, up, and down) to



(a) Maze 4 (b) Maze 5 (c) Maze 6

Fig. 4: Distribution of Global Optimal Policies in the Maze Problems. The total number of trials: 100. The higher count at fewer epochs, the better.

move forward as its action. A greater number of directions, e.g. eight, can be used but this increases the search space and more importantly makes visualisation of the results more cluttered so is difficult to interpret the effects of novel methods. Actions toward any black cell or boundary in a step would cause a collision. In these cases, the agent will stay at the same place until the next step.

Policies can be learned through iterations. As introduced above, a policy is a sequence of actions that are intended to lead the agent to achieve its target position. Each action in a policy has the maximum value of action among other three actions (see Equation 6). A global optimal policy is a policy with the minimum number of sequential actions that reaches its target. Global optimal policies, which are shown as sequences of blue arrows in Figure 3, are the objective that the XCS algorithms attempt to learn.

Experiments were conducted to test the XCSCR's ability to search global optimal policy in multistep problems. In each maze problem, 100 trials were conducted for each of XCS, the adapted XCS algorithm and the proposed algorithm. In each trial, the number of iterations (epochs) it took for an algorithm to achieve a global optimal policy for the first time is recorded (see Figure 4). In each iteration/epoch, an XCS agent takes steps to complete a maze navigation task. In each step of the interaction, the XCS agent interacts with the maze environment through an XCS iteration loop (see Section II). The XCS agent will be implemented by the XCSCR, the adapted XCS algorithm, and the standard XCS. These XCS algorithms have common parameters with the same settings from Butz [6]. Other parameters that are modified are specified in the method section.

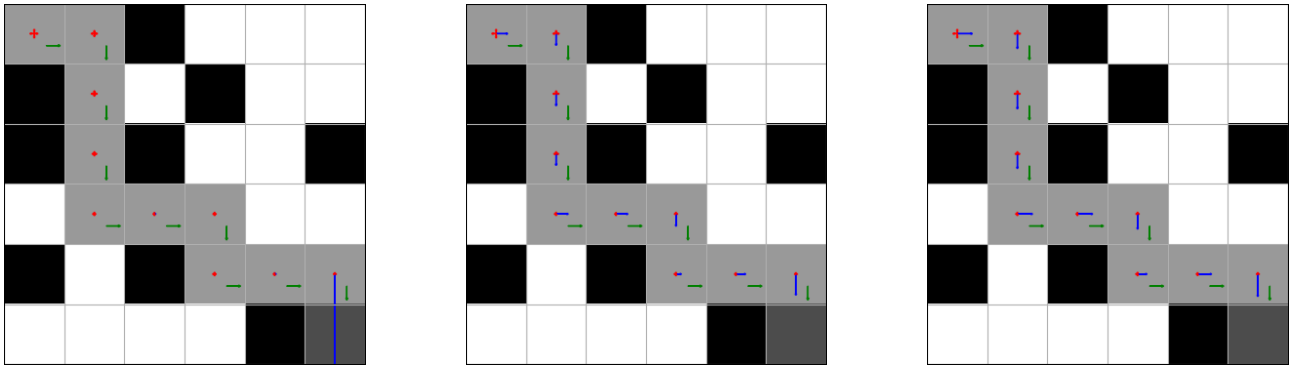
B. Results and Analysis

The XCSCR performed better than the previous adapted XCS and the standard XCS in all of the three maze environments. XCSCR achieved the global optimal policy more often than the other two algorithms (see Figure 4). In maze 4, XCSCR achieved the global optimal policy 79 times, compared to 77 times of the adapted XCS and 7 times of the standard XCS. In maze 5, XCSCR achieved 54 times, compared to 29 times of the adapted XCS and 2 times of the standard XCS. In maze 6, XCSCR achieved 65 times, compared to 25 times of the adapted XCS and zero times of the standard XCS within 180 epochs.

In addition, XCSCR achieved global optimal policy earlier than the other two algorithms in these three mazes. The distributions of the XCSCR are concentrated on early epochs, while the adapted XCS and the standard XCS distribute among the entire epoch-axis (see Figure 4). In maze 4, major achievements of XCSCR happened before the fortieth epochs, while achievements of the adapted algorithm and standard XCS were distributed evenly. In maze 5, major achievements of XCSCR happened before the seventieth epochs, while achievements of the adapted algorithm majorly were distributed after the seventieth epochs. In the maze 6, major achievements of XCSCR also happened before the seventieth epochs, while achievements of the adapted algorithm majorly were distributed after the seventieth epochs. Therefore, XCSCR changed the distribution of achievement of the global optimal policy, and XCSCR achieved the global optimal policy more frequently and earlier than the adapted XCS and the standard XCS.

Analysis of the agent's learning process illustrates the XCSCR's effects on generating policy. The short-term reward mechanism contributes to policy emerging in an early learning phase. Since the long-term, positive rewards are scarce at this phase, the worth of rules were estimated by punishments, especially through the short-term reward filter (see Figure 1.c). Thus, a policy is based on comparisons on a negative worth of rules (see Figure 5.a). In addition, magnitudes of negative worth are decreasing from the starting cell to the targeted cell in a policy, while magnitudes of positive worth are increasing. This suggests the value of the worth is related to the vastness of the searching space. The further the cell is from the targeted cell, the more vast a searching space will be, and the higher likelihood that rules receive negative rewards.

The long-term reward mechanisms tend to increase the stability of a policy as iterations progress. The long-term reward filter, which combines the three novel reward mechanisms with the standard one, can propagate long-term positive rewards to rules. For example, in the first four steps of Figure 5.b, the



(a) Iteration = 44
The policy is bootstrapped by negative rule worth during the initial learning phase.

(b) Iteration = 79
The policy is bootstrapped by positive rule worth as the training progresses.

(c) Iteration = 111
The positive worth become equivalent after sufficient iterations.

Fig. 5: Policy Learning Process in the Maze 4
Blue and red arrows: worth of rules, the direction of an arrow suggests the direction of action in the rule, magnitude suggests the value of the worth, red suggests negative worth and blue suggests positive worth. Green arrows: policy, sequential rules with the best worth, the direction of an arrow suggests the direction of the best rule, the magnitude is fixed. Light grey cells suggest they have been visited by the agent in this epoch of a trail. The dark grey cell suggests the location where the agent ended up at the end of the epoch.

values of the worth of rules, which belong to the policy, turn from negative to positive. In contrast, the values of the worth of rules that suggest a different action from the optimum policy remain negative.

As the training encounters sufficient iterations, the magnitude of the worth becomes more even. The long-term positive rewards were evenly assigned to rules in different steps. The magnitudes of the worth, which advocate for a policy, became more evenly spread in Figure 5.c than they are in Figure 5.b.

The changing worth of rules in the XCSCR learning iterations also indicates that XCSCR was able to increase the quality of policy in terms of stability. The stability of a policy depends on the difference between the worth of rules in the same state. A policy will increase its stability as the differences increase. In the experiments, the differences along a policy were increased by XCSCR learning (see Figure 5), suggesting that the quality of the policy was increasing.

V. CONCLUSION

This paper proposed XCSCR, an XCS algorithm with an combined reward method, for global optimal policies in multistep problems. The XCSCR adopts four novel mechanisms to the reward method for better usages of long-term and short-term rewards. Experiments were conducted in three standard maze environments (the maze 4, 5, and 6) to test the XCS algorithms' performances on their search global optimal policies. Results show XCSCR performed better than the standard XCS and the adapted for robotics XCS methods. In all the three mazes, the XCSCR enabled global optimal policies to emerge earlier and more frequent than the other two approaches. Analyses also provide interpretable insights about the policy learning process. The insights illustrate effects of the the four novel mechanisms, which allows XCSCR to increase the stability of the policy throughout the training iterations.

REFERENCES

[1] Minsky, M., 1961. Steps toward artificial intelligence. Proceedings of the IRE, 49(1), pp.8-30.

[2] Grefenstette, J.J., 1988. Credit assignment in rule discovery systems based on genetic algorithms. Machine Learning, 3(2-3), pp.225-245.

[3] Li, Y., 2017. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.

[4] Williams, H., Lee-Johnson, C., Browne, W.N. and Carnegie, D.A., 2015, May. Emotion inspired adaptive robotic path planning. In Evolutionary Computation (CEC), 2015 IEEE Congress on (pp. 3004-3011). IEEE.

[5] Williams, H., 2016. Human inspired robotic path planning and heterogeneous robotic mapping. PhD dissertation, Victoria University of Wellington.

[6] Butz, M.V. and Wilson, S.W., 2002. An algorithmic description of XCS. Soft Computing, 6(3-4), pp.144-153.

[7] Wilson, S.W., 1995. Classifier fitness based on accuracy. Evolutionary computation, 3(2), pp.149-175.

[8] Pomerleau, D.A., 1991. Efficient training of artificial neural networks for autonomous navigation. Neural Computation, 3(1), pp.88-97.

[9] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K. and Kumaran, D., 2016. Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673.

[10] Whitehead, S.D. and Lin, L.J., 1995. Reinforcement learning of non-Markov decision processes. Artificial Intelligence, 73(1-2), pp.271-306.

[11] Nakata, M., Lanzi, P.L. and Takadama, K., 2012, December. XCS with adaptive action mapping. In Asia-Pacific Conference on Simulated Evolution and Learning (pp. 138-147). Springer, Berlin, Heidelberg.

[12] Lanzi, P.L., 1999. An analysis of generalization in the XCS classifier system. Evolutionary Computation, 7(2), pp.125-149.

[13] Butz, M.V., Kovacs, T., Lanzi, P.L. and Wilson, S.W., 2004. Toward a theory of generalization and learning in XCS. IEEE transactions on evolutionary computation, 8(1), pp.28-46.

[14] Cheng, X., Browne, W.N. and Zhang, M., 2018, July. Decomposition Based Multi-Objective Evolutionary Algorithm in XCS for Multi-Objective Reinforcement Learning. In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-8). IEEE.