Evol. Intel.
DOI 10.1007/s12065-015-0125-y

SPECIAL ISSUE

A brief history of learning classifier systems: from CS-1 to XCS and its variants

Larry Bull

Received: 25 February 2014 / Revised: 16 October 2014 / Accepted: 11 January 2015

Abstract The direction set by Wilson's XCS is that modern Learning Classifier Systems can be characterized by their use of rule accuracy as the utility metric for the search algorithm(s) discovering useful rules. Such searching typically takes place within the restricted space of co-active rules for efficiency. This paper gives an overview of the evolution of Learning Classifier Systems up to XCS, and then of some of the subsequent developments of Wilson's algorithm to different types of learning.

Keywords Anticipation · Classification · Clustering · Function approximation · Reinforcement learning

1 Introduction

Learning Classifier Systems (LCS) are rule-based systems, where the rules are usually in the traditional production system form of "IF condition THEN action". An evolutionary algorithm and/or other heuristics are used to search the space of possible rules, whilst another learning process is used to assign utility to existing rules, thereby guiding the search for better rules. The LCS formalism was introduced by Holland [55] and based around his better known invention—the Genetic Algorithm (GA) [54]. A few years later, in collaboration with Judith Reitman, he presented the first implementation of an LCS, termed "Cognitive System Level 1" (CS-1) [60]. Holland then revised the framework to define what would become the standard system for many years [56]. However, Holland's full system was somewhat complex and practical experience found it difficult to realize the envisaged behaviour/performance, despite numerous modifications (e.g., see [119]), and interest waned. Some years later, Stewart Wilson introduced a form of LCS in which rule fitness is calculated solely by the accuracy of the predicted consequences of rule actions—XCS [113]. The following two decades have seen a resurgence in the use of LCS as XCS in particular has been found able to solve a number of well-known problems optimally (e.g., see [23]). Perhaps more importantly, LCS have been applied to a number of real-world problems (e.g., see [11]), particularly data mining (e.g., see [21]), to great effect. Formal understanding of LCS has also increased (e.g., see [18]). The purpose of this paper is to provide some historical context to the area of modern accuracy-based Learning Classifier Systems before presenting some of the main developments since the introduction of XCS 20 years ago (see [70, 105] for a previous historical reviews). The use of evolutionary algorithms to design whole rule sets, i.e., so-called Pittsburgh-style LCS [85], is not considered here.

2 The evolution of LCS

Holland developed the LCS formalism as an approach to reinforcement learning, that is, learning through trial-and-error. Reinforcement learning methods seek to ascertain the value of executing each possible action (assertion) available within each state (condition) of a given problem (see [96] for an introduction). Within psychology, the study of trial-and-error learning can be traced back to Edward Thorndike and his "Law of Effect" [99], and within computer science to Alan Turing and his "P-type unorganised machine" [104]. Farley and Clark [41] were

L. Bull (✉)
Department of Computer Science and Creative Technologies, University of the West of England, Bristol BS16 1QY, UK
e-mail: larry.bull@uwe.ac.uk

Published online: 29 January 2015

perhaps the first to implement reinforcement learning within a computer, Holland was influenced by Arthur Samuel's seminal early work on draughts/checkers [80], which itself drew upon Claude Shannon's work on chess, seemingly the first consideration of learning a value function through experience [84]. Samuel [80] described a scheme for adjusting temporally successive estimates of the end reward value from a sequence of moves (improved in [81]).

Holland's [55] interest was in how an artificial system may continuously adapt to novelty, significantly, extending previous studies by also considering how to build suitable knowledge representations thereby enabling flexible, continual learning through trial-and-error to maximise reward. A variant of his Genetic Algorithm was incorporated as an effective approach to this ability. The suggestion that a simulated evolutionary process may prove useful in artificial systems was first made by Turing [104], with early implementations within a computer by Fraser [45] and Box [10] (see [40] for an introduction). The combined reinforcement learning and evolutionary computing architecture was termed CS-1 [60]. Figure 1 shows a family tree of the LCS considered as the significant steps in the evolution of XCS from CS-1 to be discussed in this paper, beginning with CS-1.

2.1 Cognitive system Level 1

On each discrete time cycle, CS-1 receives a binary encoded description of the current state of its environment. The system determines an appropriate response based on this input, its last action, and the current contents of an internal memory space, termed a message list (Fig. 2). The rule-base consists of a population of N condition-assertion rules or "classifiers". The rule conditions are strings of characters from the ternary alphabet {0, 1, #}. The # acts as a wildcard allowing generalisation such that the rule condition 1#1 matches both the input 111 and the input 101, for example. Rule assertions contain both an action and an internal message, both formed from binary strings. All rule components are initialised randomly. Also associated with each rule are a number of parameters, including age, frequency of use, and a prediction of the typical reward received from its use, which is also the fitness metric (explained later).

On receipt of an input message, the rule-base is scanned and any rule whose condition matches the external message, the content of the message list, and the previous action becomes a member of the current "match set" [M]. A heuristic considering aspects such as the specificity of matching and the predicted future reward is then used to determine the top ten eligible rules. The system response is then chosen probabilistically from that set. The chosen rule's action is executed in the environment, the message list updated, its age halved, and frequency increased.

CS-1 uses an epochal reinforcement learning scheme such that the identification of all rules that have provided an action is recorded, in order, between rewards. If an external reward is received, the predicted reward of each

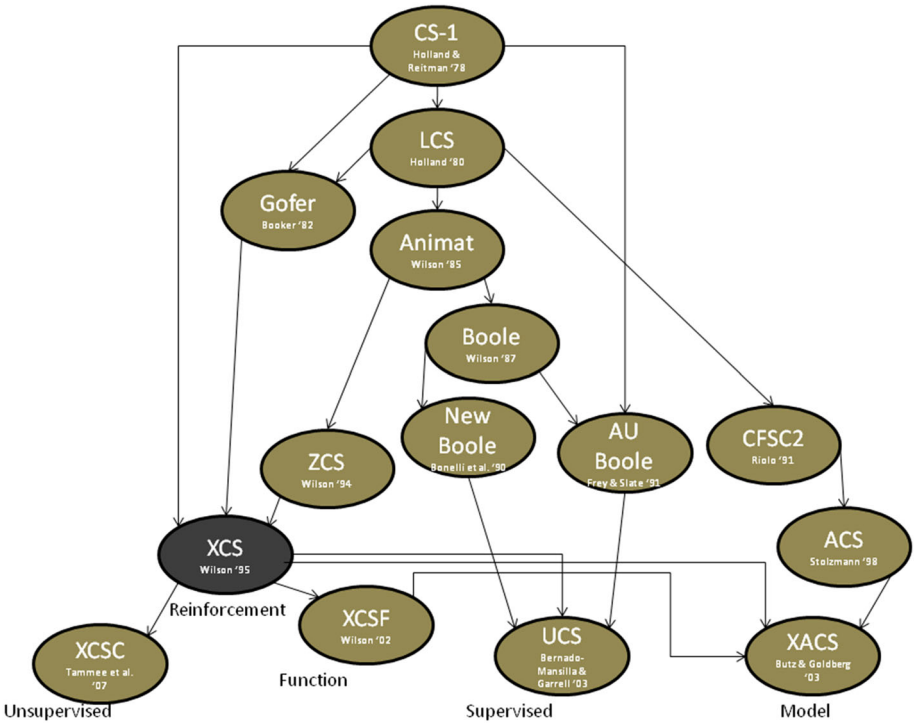

Fig. 1 Learning classifier systems' family tree

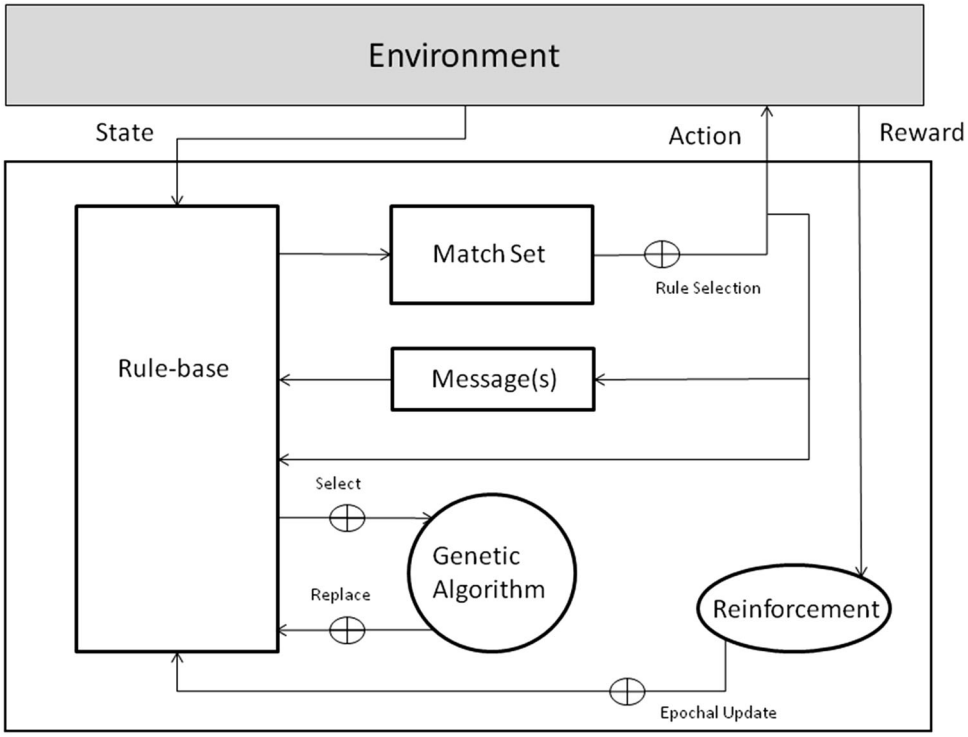

Fig. 2 Schematic of CS-1

rule in this set [E] is adjusted at a rate inversely proportional to their frequency parameter "to reflect their accuracy in anticipating this reward. Those predicted payoffs that were consistent with (not greater than) this reward are maintained or increased; those that overpredicted are significantly reduced" [60]. A further heuristic is applied to the predictions such that the actual value of reward used to update each member of [E] is "attenuated", an adjustment based on the relative size of reward predicted by rules and by their successors in [E], it being incremented each time the latter is higher than the former. "This parameter is highly correlated with the delay between a response and the reward" [60] and may be seen as an early form of temporal difference learning.

After every ten rewards received, the contents of the rule-base are altered by the simulated evolutionary process of the GA. The implemented CS-1 could take one of two actions and so ran the evolutionary search process within the two sub-populations, that is, action niches. Fitness proportionate selection using the predicted reward of each rule as the fitness value picks two parents from the rule-base population. These are then combined using one-point crossover (mutation is not mentioned). One of the two offspring produced is selected at random to be inserted into the rule-base. Replacement uses the age of rules. "Recall that a classifier with a poor predicted payoff rarely wins competitions; without a win, its age increases steadily. Age therefore, reflects the classifier's quality as well as its frequency of use. To make room for the new classifier therefore, one with an old age is deleted." [60]. Moreover, from the set of oldest rules within the niche, the one closest in Hamming(-like) distance is chosen; a form of crowding is used.

CS-1 was shown able to solve a simple maze task with seven locations, two actions, and two types of reward, before being applied to an extended maze. Holland and Reitman report faster learning of the second maze using a system previously trained on the smaller maze, in comparison to a naïve system. Analysis of the external input patterns indicates minor changes in effective general rules in the smaller maze are close in rule-space to those in the larger, as might be expected (see [66] for related recent work).

LCS aim to build an efficient representation of any underlying regularities within the given problem domain during learning. The inherent pressure within CS-1 to discover *maximally general* rules—rules which aggregate the most problem states together from which the same action results in the same reward—over more specific (less # symbols in their condition) but equally accurate predictors comes from the evolutionary deletion scheme. More specific rules tend not to match so often and so their age increases more rapidly than more general, but also accurate rules; the probability of removal of specific rules from the rule-base increases with specificity. Similarly, the pressure to remove *over general* rules—rules that aggregate too many states together such that the level of reward received from their use varies—comes from the reinforcement scheme and evolutionary selection. Over or under prediction of a reward value results in a significant reduction in the predicted reward of a

rule, the parameter used as the fitness measure for reproduction by the GA; inaccurate rules have lower fitness and hence are unlikely to be selected.

However, the described system struggles to maintain more than one or a very few rules within a population. That is, the GA tends to converge upon a single (maximally general) solution. This explains why CS-1 runs the GA in the two explicit action sub-population niches. In the two mazes, CS-1 always started in the middle and had to maintain the same action across a number of states to an end goal state where reward was given. Whether the system should go left or right depended upon an internal value. Hence a rule which generalised over all the states to the left and one which generalised over all the states to the right was the optimal solution. By running the GA in two niches based on actions, both were sustainable indefinitely.

With this apparent limitation, Holland subsequently altered a number of aspects of CS-1 in his "standard system", most notably removing the use of reward prediction accuracy and frequency of rule use. CS-1's use of both in part anticipated XCS's dependence on similar qualities.

2.2 Holland's standard architecture

A few years later Holland [56] revised CS-1 and described what would become the standard architecture, here termed "Learning Classifier System" (LCS). It should be noted that Holland seems not to have used the prefix "learning" at the time; Goldberg [49] may have been first to add the emphasis. The main change from CS-1 was to introduce a reinforcement learning scheme based upon an economic metaphor, known as the "bucket brigade" (after the water passing chains of fire fighters), in which rule utility is judged by the accrual of credit. In this way, rules acting in temporal chains leading to external reward are viewed as the middlemen of supply and demand chains. Rules maintain a single parameter of credit (termed strength) received. This is used both for action selection and in rule discovery by the GA. The message list is extended to enable multiple rules to post their assertions. Rule conditions no longer have a fixed structure to consider the current environmental state, the contents of the message list and the last action. Instead, all conditions and assertions are of the same length, with conditions also able to include a logical NOT. Assertions are now built from the same alphabet as conditions {0, 1, #} such that information may "pass through" from either the condition or the string (external input or internal message) which the rule matches where a # exists.

On each cycle, a binary external state description is placed onto the message list, the rule-base is scanned, and any rule whose condition matches the external message and/or the other contents of the message list becomes a member of [M] (Fig. 3). Rules are selected from those comprising [M], through a bidding mechanism, firstly to become the system's external action and then to post their assertion onto the (fixed size) message list for the next cycle. This selection is performed by the roulette wheel scheme based on rule bids. Rules' bids consist of two components, their strength and the specificity (fraction of non-# bits) of their condition. Further, a constant (here termed β, where $0 < \beta < 1$) is factored in, i.e., for a rule C in [M] at time t:

$$\text{Bid}(C,t) = \beta \cdot \text{specificity}(C) \cdot \text{strength}(C,t)$$

Reinforcement consists of redistributing bids made between subsequently chosen rules. The bid of each winner at each time-step is placed in a "bucket". A record is kept of the winners on the previous time step and they each receive an equal share of the contents of the current bucket; fitness is shared amongst concurrently activated rules. If a reward is received from the environment then this is paid to the winning rule which produced the last system output. However whether all rules that have posted a message share the external reward appears to vary in the literature, being both included [57] and excluded [58]. "Thus, the bucket brigade assures that early-acting, stage-setting classifiers receive credit if they (on average) make possible later, overtly rewarding acts" [58]. The emphasis upon average ability relaxes the previous explicit focus on accurately predicting reward; there is an apparent reduction in the selective pressure for consistent behaviour which formed the basis of CS-1. With hindsight, this change was perhaps the most significant between the two LCS.

As noted above, the periodically applied GA uses rule strength to select two parents, these are then combined using one-point crossover and mutated. Both offspring are inserted into the rule-base, replacing rules chosen inversely proportional to their strength. Since reward is shared amongst rules, the GA is in principal able to maintain multiple useful rules within the rule-base (discussed later).

A number of other mechanisms were proposed by Holland but for the sake of clarity they are not described in detail here. These include the idea that hierarchical rule associations could emerge via specific rules out-bidding more general rules in certain important situations, and extra "tag" regions of conditions and assertions being added would aid the formation of sequential induction (see [61] for a full treatment). These ideas do not feature in modern LCS (see [87] for an exception).

2.3 GOFER

Booker [6] presented a form of Holland's standard LCS which extends the principle of using a GA to discover any underlying regularities in the problem space, dividing the task of learning such structure from that of supplying appropriate actions to receive external reward (see [8] for an overview]). Here a separate LCS exists

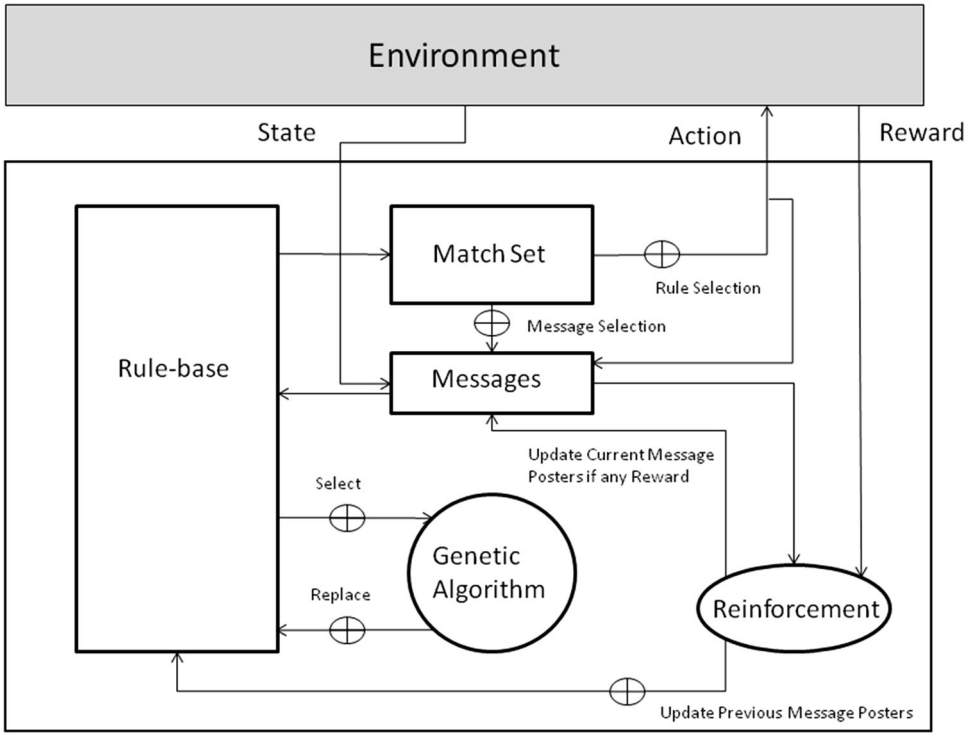

Fig. 3 Schematic of Holland's LCS

for each of these two aspects. A first LCS receives binary encoded descriptions of the external environment, with the objective to learn appropriate regularities through generalizations over the "perception" space. This is seen as analogous to learning to represent categories of objects. The matching rules not only post their messages onto their own message list but some are passed as inputs to a second LCS. The second LCS therefore only receives reward when it correctly exploits such categorizations with respect to the current task. GOFER contains a number of innovations including partial matching and rule excitation levels, however it is the use of restricting the process of rule-discovery to concurrently active rules which has proven most influential (see [7]). Here parents are chosen from within a given [M] thereby avoiding the mixing of rules with generalizations which (potentially) consider markedly different aspects of the problem. Booker [9] later extended the idea to trigger the GA during learning whilst also leaving it running at a constant rate under the reinforcement process as Holland did. In particular, rules maintain an approximation of their "consistency", a measure of the variance in the reward they receive. If a given percentage of rules in [M] have a level of inconsistency above a threshold, the fitness of all consistent rules is increased and the GA run: "… consistent classifiers are thereby made more attractive to the genetic algorithm" [ibid.]. XCS uses both a form of triggered niche GA and rule consistency.

2.4 ANIMAT and ZCS

Stewart Wilson began to develop versions of Holland's LCS as an approach to understanding animal/human intelligence through the computer simulation of simple agents in progressively more complex domains—termed the animat approach. The first of these, ANIMAT [110], makes a number of simplifications to Holland's architecture. In particular, the message list is removed and matching rules are grouped by their action in the bucket brigade process, forming action sets [A]. The GA is also sensitive to rule actions, somewhat akin to CS-1: a first parent is chosen based upon its strength from the rule-base, the second is then chosen from the subset of the population with the same action. ANIMAT controlled a simple agent in a 2D grid-world, able to sense the contents of the eight locations surrounding it and able to move in each such direction if clear. Wilson showed learning was possible such that effective paths to food reward signals were discovered. However, he noted that the system had "nothing which preferentially reinforces the most expeditious classifiers" [ibid.]. To encourage the shortest path to reward from a given start location, rules were extended to maintain an estimate of the number of subsequent steps to reward from their use, updated locally based upon the estimates of successor rules. This was factored into action selection via dividing strength by distance. ANIMAT also includes a guided recombination operator, replacing dissimilar bits in parent conditions with a # to aid the formation of useful generalizations.

Wilson later returned to ANIMAT, further simplifying it in his "zeroth-level" classifier system (ZCS) [112] (Fig. 4). Importantly, the bucket brigade was again modified to incorporate a mechanism from temporal difference learning [95] (see also [37] for an early connection). Here the fraction of the total strength of a given [A] in the bucket is further reduced by a discount rate γ $(0 < \gamma < 1)$ before being shared equally amongst the rules of the previous action set $[A]_{-1}$. Discounting allows systematic control over the influence of future rewards, replacing Wilson's previous distance approximation mechanism. The effective update of action sets is thus $(0 < \beta < 1)$:

$$\text{strength}([\text{A}], t+1) = \text{strength}([A], t) + \beta \cdot [\text{Reward} + \gamma \cdot \text{strength}([\text{A}]_{+1}) - \text{strength}([\text{A}], t)]$$

To give increased focus to the search, rules in a given [M] but not [A] have their strengths reduced by a tax; rules can only persist if they regularly receive (high) reward. A "create" mechanism in ANIMAT is also retained in ZCS, but slightly modified. Here, if an [M] is empty or if the total strength of [M] is below a given threshold, a new rule is created to cover the current environmental input, randomly augmented with some #, and given a random action. The action niche restriction and generalization mechanisms of the GA are removed. Parental rules give half of their strength to their offspring under the GA which fires at a fixed rate ρ.

Results with ZCS indicated it was capable of good, but not optimal, performance [112, 33]. Wilson [112] also included a version of the off-policy temporal difference learning algorithm Q-learning [109] to some benefit. He also proposed to use the triggered niche GA of GOFER on top of the panmictic/global scheme described above. Bull [12] showed the potential for disruption of the reward sharing scheme using just a niche GA in a similar LCS but no combination is known. It has been shown that ZCS is capable of optimal performance in a number of well-known test problems but that it appears to be particularly sensitive to some of its parameters [16], and it has been shown to outperform XCS in classes of noisy domain [92]. XCS maintains a number of ZCS's basic features but makes significant alterations.

2.5 BOOLE, NEWBOOLE and AU-BOOLE

After introducing a number of modifications to Holland's architecture in ANIMAT, but before ZCS, Wilson presented a specialised form designed for reinforcement learning tasks where immediate reward is given. In particular, his BOOLE system was designed for binary decision tasks [111]. BOOLE maintains the [M] → [A] mechanism of ANIMAT, also removing the message list. The GA no longer restricts selection of the second parent to having the same action when using crossover, and reproduction causes the strength of parents to be reduced and donated to offspring akin to the mechanism later used in ZCS. It has been shown that reducing strength can create a pressure for more general rules as they update more frequently and therefore regain reward faster [12]. Again, as in the later ZCS, rules in [M] but not

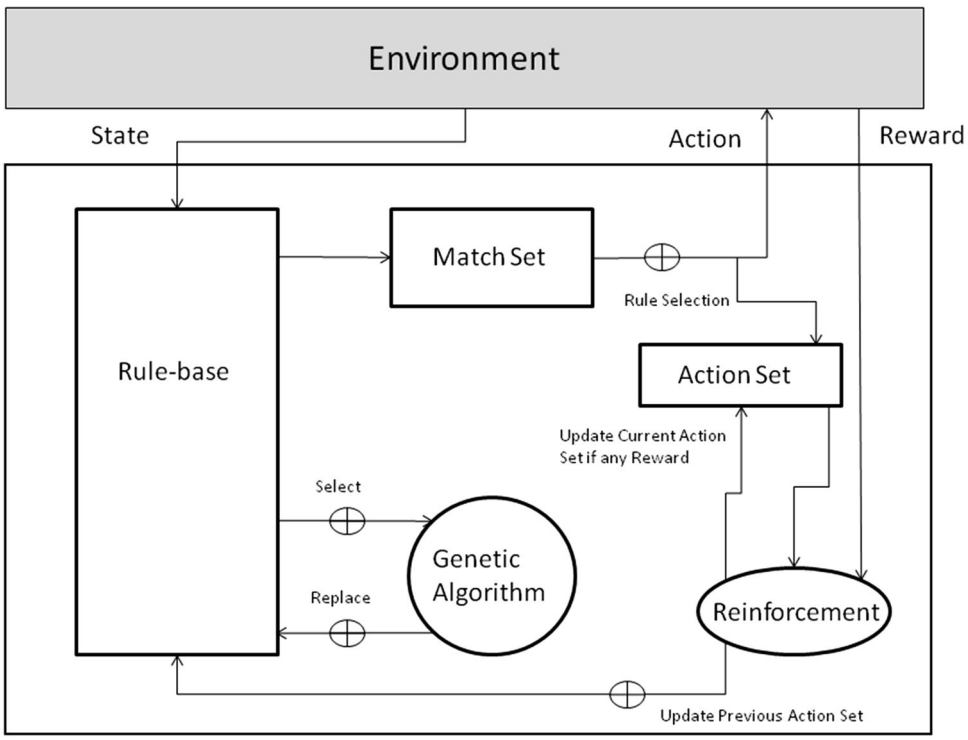

Fig. 4 Schematic of Wilson's ZCS

[A] have their strengths reduced by a tax. BOOLE was shown able to learn two- and three-address bit multiplexer problems (6MUX and 11MUX, respectively), with the effects of varying the tax rates, genetic operators and including a reward bias based upon the degree of generalisation in rules explored.

Bonelli et al. [5] made the significant step of presenting a form of LCS for supervised learning tasks, that is, tasks where the correct response is known at the point of internal updating. Extending BOOLE, they noted that the set of rules in [M] providing the correct response, regardless of whether they formed [A], should receive reward. Hence they split [M] into the correct set [C] and incorrect set Not[C] for their NEWBOOLE system. BOOLE's uses of taxes and a bias in the distribution of reward based upon generality were kept. They showed significant improvement in learning speed compared to BOOLE and to an artificial neural network using backpropagation on the 6MUX and 11MUX tasks. Hartley [51] showed NEWBOOLE to be competitive with XCS on a well-known set of binary classification tasks, although XCS's maintenance of a full state-action-reward map gave it an advantage in some forms of non-stationary task (see [16] for discussion).

Seemingly independently, Frey and Slate [46] also presented a variant of BOOLE for supervised learning tasks in which they also update the correct set within [M] regardless of the output. Having struggled to find the correct balance of taxing and bid biasing for a letter recognition task, with reference to Holland's [55] original ideas, they introduced the accuracy-utility system (here termed AU-BOOLE). Here each rule maintains two parameters: accuracy, the ratio of correct bids to total bids made; and, utility, the ratio of correct bids when chosen to total number of times chosen as the output. Accuracy is used in bidding in [M] and for reproduction, and utility is used for deletion. Whilst performance with AU-BOOLE was found to be similar to their version of NEWBOOLE, they report greater ease in finding useful parameters. As will be discussed, these ideas have been incorporated into XCS, resulting in the "sUpervised Classifier System" (UCS) [3].

2.6 CFSC2 and ACS

Holland and Reitman [60] suggested a number of extensions to CS-1 at the end of their paper, particularly ways by which to learn more sophisticated cognitive maps than the stimulus–response relations they had achieved. "Cognitive maps allow the system to use lookahead to explore, without overt acts, the consequences of various courses of action." [ibid.]. Again, following Samuel [80], they describe a scenario of rules being linked over system cycles through the message list which do not cause external actions on each step. Holland [59] later returned to this aspect, proposing that the aforementioned extra "tag" regions of conditions and assertions that can be added as arbitrary patterns would aid the formation of sequential induction of the necessary form: IF condition AND assertion THEN next-condition. That is, Holland did not seek to change the rule structure from his standard LCS to this direct form. Riolo [78] was first to implement lookahead capabilities within LCS with his CFSC2. He allowed the system to execute more than one cycle before providing an action, added tags along the lines Holland [59] had suggested, and introduced an extra strength parameter to represent the predictive accuracy of a rule. Through tags, rules are either connected to external or internal events, or both. Bidding is adjusted to also factor the accuracy of predicted next states of a rule (if any). Rule chains which accurately map features in simple mazes with or without overt reward (latent learning) are reported to emerge under a rule discovery process which is driven by internal and external messages rather than a GA. That is, when no rules match or none are chained across system cycles, various heuristics are used to form appropriate rules via tags. Roberts [79] presented a related approach within ANIMAT which maintained "followsets", time-stamped information regarding rewards received or next states obtained after a rule had fired. The value of such rewards is factored into rule strengths.

Wilson [113] proposed altering the rule structure to contain the anticipated next state, with an "expecton" in XCS. Stolzmann [90] presented a system in which such a rule structure is used (the expecton component termed the "effect")—the Anticipatory Classifier System (ACS). Drawing upon a learning theory from cognitive psychology, sub-populations of rules are learned per [A] via a specialisation heuristic (not a GA) for rules based upon the environmental input, both in the condition and effect components. Rule utility is represented by the accuracy of anticipations whilst external reward is used in bidding. A famous experiment with rats in a T-maze [83] is simulated and the results indicate similar behaviour from the ACS. A combination of ACS and XCS has been presented to achieve such model learning, as will be discussed (e.g., see [25]).

3 Wilson's XCS

The most significant difference between XCS (Fig. 5) and all other LCS prior to its presentation is that rule fitness for the GA is not based on the amount of reward received by rules but purely upon the accuracy of predictions (p) of reward. The intention in XCS is to form a complete and accurate mapping of the problem space (rather than simply focusing on the higher payoff niches in the environment) through efficient generalizations: XCS learns a value function over the complete state-action space. On each

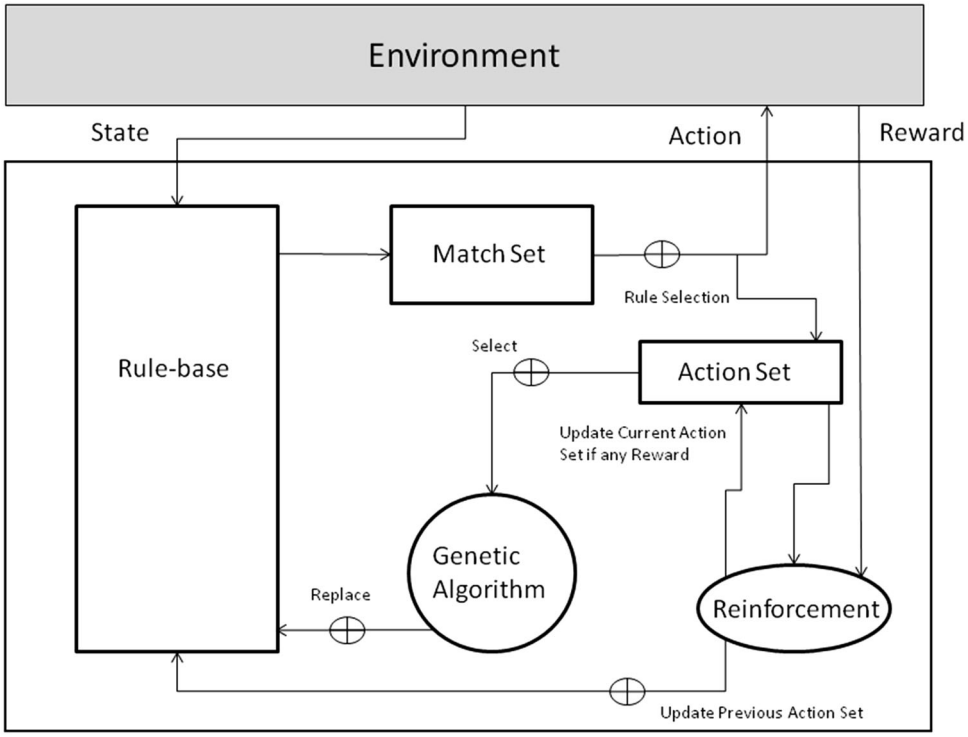

Fig. 5 Schematic of Wilson's XCS

time step a match set is created. A system prediction is then formed for each action in [M] according to a fitness-weighted average of the predictions of rules in each [A]. The system action is then selected either deterministically or randomly (usually 0.5 probability per trial). If [M] is empty covering is used.

Fitness reinforcement in XCS consists of updating three parameters, ε, p and F for each appropriate rule; the fitness is updated according to the relative accuracy of the rule within the set in five steps:

1. Each rule's error is updated: $\varepsilon_j = \varepsilon_j + \beta$ (| *Reward*-p_j | $- \varepsilon_j$)
2. Rule predictions are then updated: $p_j = p_j + \beta$ (*Reward*-p_j)
3. Each rule's accuracy κ_j is determined: $\kappa_j = \alpha(\varepsilon_0/\varepsilon)^{\nu}$ or $\kappa = 1$ where $\varepsilon < \varepsilon_0$ where ν, α and ε_0 are constants controlling the shape of the accuracy function.
4. A relative accuracy $\kappa_j^{'}$ is determined for each rule by dividing its accuracy by the total of the accuracies in the action set.
5. The relative accuracy is then used to adjust the classifier's fitness F_j using the moyenne adaptive modifee (MAM) [107] procedure: If the fitness has been adjusted $1/\beta$ times, $F_j = F_j + \beta(\kappa_j^{'} - F_j)$. Otherwise F_j is set to the average of the values of $\kappa^{'}$ seen so far.

In short, in XCS fitness is an inverse function of the error in reward prediction, with errors below ε_0 not increasing fitness. The maximum $P(a_i)$ of the system's prediction array is discounted by a factor γ and used to update rules from the previous time step. Thus XCS exploits a form of Q-learning [109] in its reinforcement procedure. The GA originally occurred in [M] but Wilson [114] later move it to [A] to further reduce the potential for recombining rules inappropriately, i.e., when there is significant asymmetry in the generalisation space for each action in a given match set (see [15] for discussion). Two rules are selected based on fitness from within the chosen [A]. Rule replacement is global and based on the estimated size of each action set a rule participates in with the aim of balancing resources across niches. The GA is triggered within a given action set based on the average time since the members of the niche last participated in a GA (after [9]). See [24] for a full algorithmic description of XCS.

Wilson originally demonstrated results on multiplexer functions and a maze problem. Importantly, he shows how maximally general solutions are evolved by XCS. This is explained by his "generalization hypothesis":

> Consider two classifiers C1 and C2 having the same action, where C2's condition is a generalization of C1's. … Suppose C1 and C2 are equally accurate in that their values of ε are the same. Whenever C1 and C2 occur in the same action set, their fitness values will be updated by the same amounts. However, since C2 is a generalization of C1, it will tend to occur in more [niches] than C1. Since the GA occurs in [niches], C2 would have more reproductive

> opportunities and thus its number of exemplars would tend to grow with respect to C1's. … C2 would displace C1 from the population" [113].

Butz et al. [26] studied this hypothesis formally, introducing the concept of different pressures acting within XCS and then examined how they interact. They term the process described by Wilson above as the set pressure, which occurs due to the niche GA for reproduction and global GA for deletion. Kovacs [67] extended Wilson's idea, presenting the "optimality hypothesis" which suggests that due to the set pressure, XCS has the potential to evolve a complete, accurate and maximally general (compact) description of a state-action-reward space. Butz et al. [26] begin by approximating the average specificity of an action set s([A]) given the average specificity in the population s([P]):

$$s([A]) = s([P])/(2 - s([P]))$$

For an initially random population, this indicates that the average specificity of a given [A] is lower than that of the population [P]. Opposing the set pressure are the pressures due to fitness and mutation since the former represses the reproduction of inaccurate overgeneral rules and the latter increases specificity. They then extend the set pressure definition to include the action of mutation, resulting in the "specificity equation":

$$s([P(t+1)]) = s([P(t)]) + f_{ga}((2 \cdot (s([A]) + \delta_{mut} - s([P(t)]))/N)$$

where δ_{mut} is the average change in specificity between a parent rule (*cl*) of specificity *s(cl)* and its offspring under mutation, defined as $0.5\mu(2–3\ s(cl))$, and f_{ga} is the frequency of GA application per cycle. It is shown that, for a number of simple scenarios such as a random Boolean function, this equation is a good predictor of resulting specificity and they note this "represent[s] the first theoretical confirmation of Wilson's generalization hypothesis" [ibid.]. The ability of XCS to maintain niches was explored formally in Butz et al. [30].

Butz et al. [26] also identified two potentially conflicting challenges for XCS, namely that the population of rules needs to be sufficiently general to cover the input space, whilst rules must be specific enough such that there is an effective fitness gradient towards accuracy. Butz [23] later showed how, by giving consideration to the bounds of these challenges, together with those of reproduction and niche support, XCS can PAC-learn a sub-class of *k-DNF* problems, i.e., learn their correct solution in polynomial time with high probability.

Since its introduction, a number of aspects from the wider field of machine learning have been explored within XCS (see [68] for a general review). From evolutionary computing, techniques such as rank-based selection (e.g., [27]), parameter self-adaptation (e.g., [64]), local search (e.g., [120]), and estimation of distribution algorithms (e.g., [29]) have been explored, along with non-binary representation schemes. The conditions in XCS have been represented by things such as real-valued intervals (e.g., see [91] for discussions), trees (e.g., [69], after a proposal in Wilson [112]) and developmental approaches [118]. The actions of rules have been represented by trees (e.g., [65], after [1]) and linear approximators (e.g., [103], after [116]). A whole rule in XCS has also been represented using fuzzy logic (e.g., [32], after [106]), neural networks (e.g., [17]), and logic networks (e.g., [13]). Techniques considered from reinforcement learning include gradient descent [28] and eligibility traces [38]. Moreover, general ideas such as the use of ensembles (e.g., [19, 20]) and multi-agent systems (e.g., [52]) have also been considered with XCS.

Wilson removed the message list from LCS in his ANIMAT and didn't return to the concept until he presented ZCS. As a possible area of future research, Wilson [112] describes a "memory register". Here a global internal register's current content/state would be matched by a defined part of each rule's condition, along with the external stimulus. Similarly, rule actions would contain an element to update the content of the register, as well as supply the external response. As such, this is very similar to the original structure of CS-1. Lanzi and Wilson [71] showed it was possible to solve non-Markov mazes through the development of the idea in XCS. An alternative approach to memory was suggested by [119] wherein rules link together to form "corporations". Tomlinson and Bull [102] showed some success with the idea in XCS.

Wilson prophetically suggests at the end of his paper introducing XCS: "The results point to the conclusion that accuracy-based fitness and a niche GA form a promising foundation for future classifier system research" [113]. As mentioned above and shown in Fig. 1, XCS and these key features have been extended from reinforcement learning as will now be discussed.

4 The evolution of XCS

4.1 UCS: Supervised Learning

Starting with NEWBOOLE, it has long been noted that in the use of LCS for tasks where there is an immediate reward indicating correctness, the standard reinforcement learning approach can be altered. UCS [3] uses the accuracy calculation of AU-BOOLE to replace the standard running average error update in XCS. That is, κ_j = number of correct classifications/experience, with experience defines as the number of times a rule has matched.

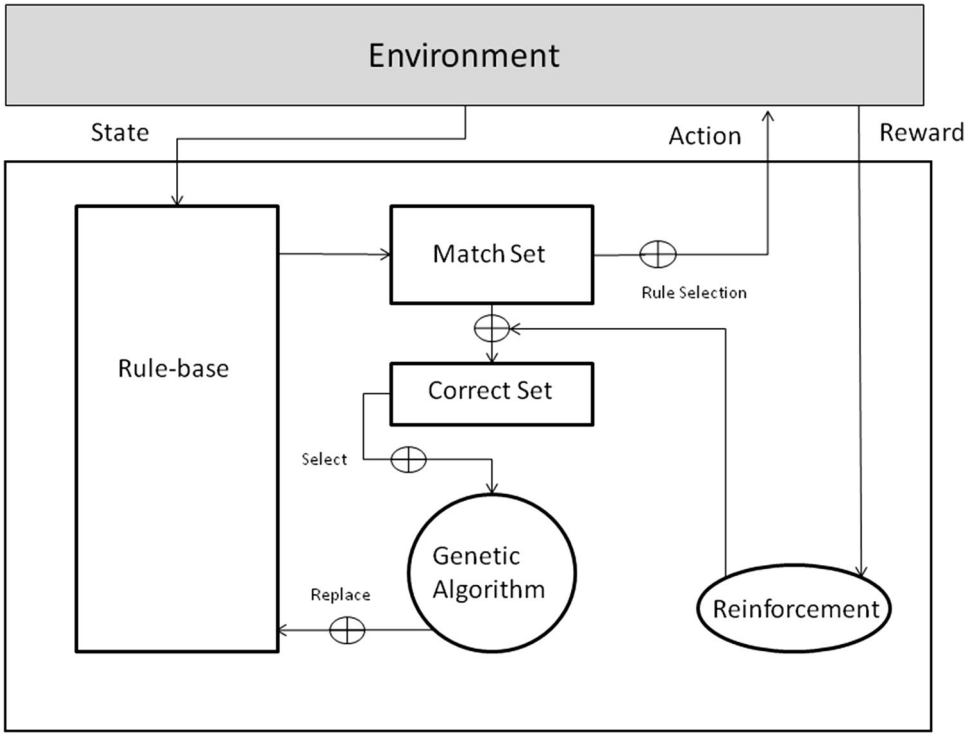

Fig. 6 Schematic of UCS

Thereafter, $F_j = (\kappa_j)^\nu$ and the GA is run in the correct set [C], with deletion a global operation based upon the size of [C] (Fig. 6).

The main effect of the change to a supervised update is that UCS only maintains a set of rules which receive high payoff, as opposed to XCS's construction of a full state-action-reward map. As a consequence, UCS was shown to learn more quickly than an equivalent XCS on a number of benchmark tasks. As well as the reduced generalization task, it was also shown to learn more effectively due to the change in the fitness pressure for certain types of problem from the simplified fitness function. UCS and XCS are shown to be competitive with a number of well-known machine learning techniques over well-known real-world datasets.

In a few cases, XCS was found to outperform UCS on the real-world datasets and it was speculated this is due to in part to a lack of fitness sharing within niches. Later inclusion of the same relative accuracy calculation into UCS gives improved performance, particularly with unbalanced datasets [74]. However, XCS remains a robust classification data mining algorithm (see [43]).

Like XCS, a number of techniques have been incorporated into UCS, such as the use of rank-based selection (e.g., [74]) and fuzzy logic (e.g., [75]). It has also been used within ensembles, including with the use of neural networks to provide the action (e.g., [35]). Ideas from the wider ensemble/mixture-of-experts literature have also been used to understand and refine UCS (e.g., [39]).

4.2 XCSC: unsupervised learning

Unsupervised learning describes those tasks under which structure is sought in unlabelled data without further external input. Perhaps somewhat surprisingly, no previous suggestion of the use of LCS for such learning is known in the literature until the work of [97, 98] on clustering (see Fig. 7). Clustering is an important unsupervised learning technique where a set of data are grouped into clusters in such a way that data in the same cluster are similar in some sense and data in different clusters are dissimilar in the same sense (see [121] for an overview). Most clustering algorithms require the user to provide the number of clusters, and the user in general has no idea about the number of clusters (e.g., see [100]). Hence this typically results in the need to make several clustering trials with different numbers of clusters from 1 to the square-root of the number of data points, and select the best clustering among the partitioning with different number of clusters.

Tammee et al. show how the generalization mechanisms of XCS can be used to identify clusters – both their number and description. Rules in their XCSC use an interval representation of the form $\{\{c_1,s_1\}, \ldots \{c_d,s_d\}\}$, where c is the interval's range centre from [0.0,1.0] and s is the "spread" from that centre from the range $(0.0,s_0]$ and d is a number of dimensions. Each interval predicates' upper and lower bounds are calculated as follows: $[c_i - s_i,\ c_i + s_i]$. If an interval predicate goes outside the problem space bounds, it is truncated. Rule fitness consists of updating the matching error ε which is derived from the

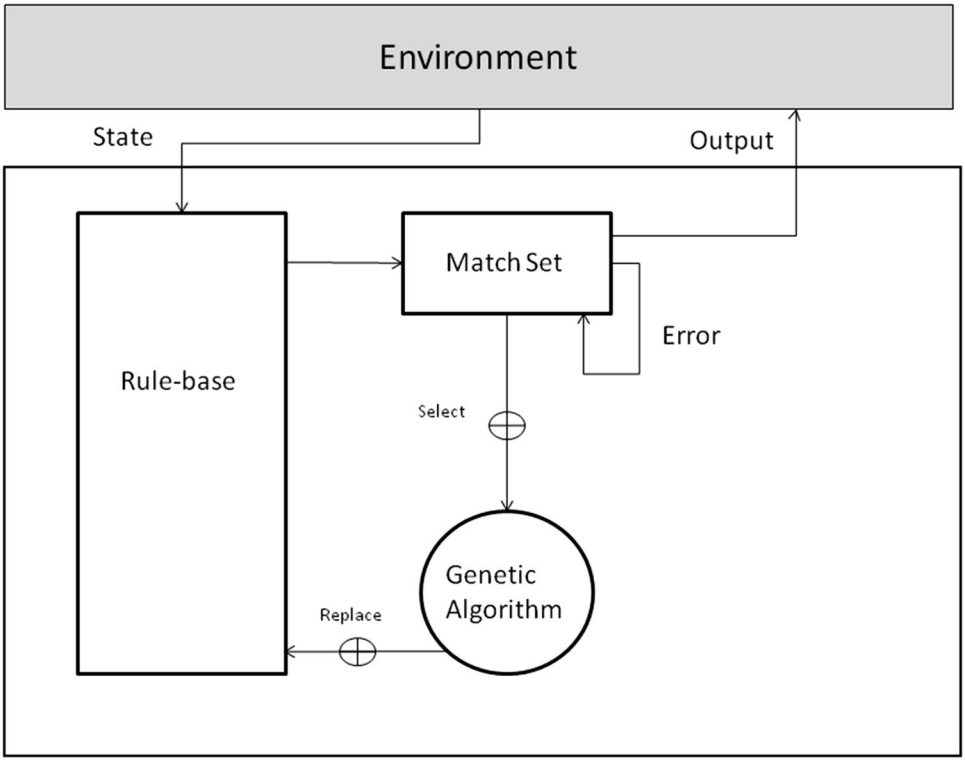

Fig. 7 Schematic of XCSC

Euclidean distance with respect to the input x and c in the condition of each member of the current [M] using the Widrow-Hoff delta rule with learning rate β:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta\left(\left(\left(\sum_{l=1}^{d}(x_l - c_{lj})^2\right)\right)^{1/2} - \varepsilon_j\right)$$

The rest of XCS processing remains unchanged. Hence the set pressure encourages the evolution of rules which cover many data points and the fitness pressure acts as a limit upon the separation of such data points, i.e., the error.

Tammee et al. [97] began by using a slightly simplified version of XCS as the underlying LCS (YCS) [12], but found that XCS's relative accuracy fitness function was more effective than a function directly inversely proportional to error [98]. Note this is similar to the aforementioned findings with UCS [74]. Moreover, since theε_0 parameter controls the error threshold of rules, Tammee et al. investigated the sensitivity of XCSC to its value by varying it. Their experiments show that, if $\varepsilon_0$is set high, e.g., 0.1, in less-separated data the contiguous clusters are covered by the same rules. They therefore developed an adaptive threshold parameter scheme which uses the average error of the current [M]:

$$\varepsilon_0 = \tau\left(\sum \varepsilon_j / N_{[M]}\right)$$

where ε_j is the average error of each rule in the current match set and $N_{[M]}$ is the number of rules in the current match set. This is applied before the fitness function calculations. Experimentally Tammee et al. found $\tau = 1.2$ was most effective for the problems they considered.

This work has recently been extended to include hierarchical cluster/rule merging and voting (e.g., [77]).

4.3 XCSF: function learning

Wilson [113] proposed that XCS could be modified to learn functions, i.e., problems of the general form $y = f(x)$, and subsequently presented XCSF [115] [116]. Rules in XCSF typically use an interval representation of the form $\{\{l_1\ u_1\}, \ldots \{l_d,u_d\}\}$, where li ("lower") and ui ("upper") are integers. A rule matches an input x with attributes xi if and only if $li \leq xi \leq ui$ for all xi. Having first used the standard prediction creation of XCS, Wilson introduces piecewise-linear approximators to each rule, i.e., functions of the form $h(x) = w_0 + w_1x_1 + \ldots w_dx_d$. Rather than add the weights w_j to the rule representation to be learned under the GA, a variant of a simple gradient descent method is employed: $\Delta w_j = (\eta/|x'|^2)\ (t - o)x_j$, where t is the target, o is the output and η is a learning rate.

Wilson shows the basic XCSF learning a sine function and multi-dimension root-mean-squared functions wherein the local approximations are shown to be adaptive to the function being approximated such that the size of the local interval responds to the curvature of the function. This increases efficiency and as well introduces an additional

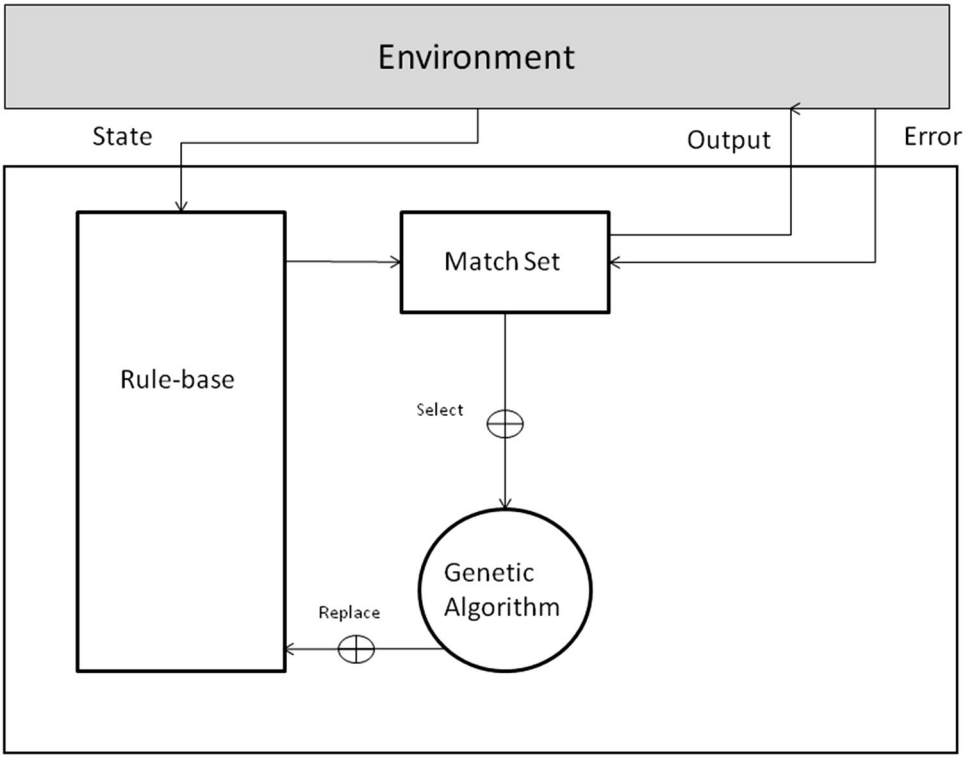

Fig. 8 Schematic of XCSF

kind of generalization. This has subsequently been explored extensively, using a variety of rule condition representations and function approximation techniques (e.g., see [31, 72]). The theoretical underpinnings of XCS have also been extended to XCSF (e.g., [88]) (Fig. 8).

Wilson [116] extended the idea to propose a generalized rule format such that the prediction associated with a state-action rule under reinforcement leaning is computed in the same way, as opposed to maintained as (an adjusted) parameter (see also [48]). Again, this has been explored using a variety of rule representations and approximators (e.g., see [73]). It can be noted that in rule representations which can also provide memory through individual rule-internal structures, such as recurrent connections in a network, this opens up new ways by which to solve non-Markov tasks (e.g., see [76]).

4.4 XACS: model learning

As noted above, Stolzmann [90] presented an accuracy-based LCS in which rules are extended to predict the subsequent sensory state from their use. That is, rules are of the general form "IF condition AND assertion THEN effect". The mechanism through which such rules are learned is based upon the theory of Anticipatory Behavioural Control [53] and not a simulated evolutionary process. The search algorithm, termed the Anticipatory Learning Process (ALP), has thus far relied upon the traditional ternary alphabet {0,1,#} (see [47] for related systems). Butz et al. introduced the use of a niched GA alongside the ALP to improve the generalization abilities of ACS, termed ACS2 (see [22] for full details). Whilst effective, ACS2 was found to sometimes struggle to form both accurate environment models and state-action-reward models simultaneously. Drawing on XCSF, ACS2 was subsequently extended in XACS [25] (Fig. 9).

XACS maintains the principle features of ACS(2), using the ALP to specialize rules when their anticipated effect does not match the next state. As in ACS2, a # symbol in an effect indicates that the bit is not anticipated to change in the next state, whereas defined bits are anticipated to change to that value (unlike ACS). The GA is the same as in ACS2, using the time triggered scheme of XCS, with the mutation process only introducing #'s and crossover only happening over conditions. Running alongside ACS2 is a variant of XCSF to learn the value of states. The rules consist of a condition and prediction parameter, as in the first version of XCSF described above. The XCSF component is used each time external reward is received from the environment, updating predicted values using both its own current prediction and the model knowledge of the ACS2 component.

Butz and Goldberg [25] report improved performance over ACS2 in blocks world problems of varying sizes. Benefits of the model include a mechanism through which to bias action selection such that those rules whose

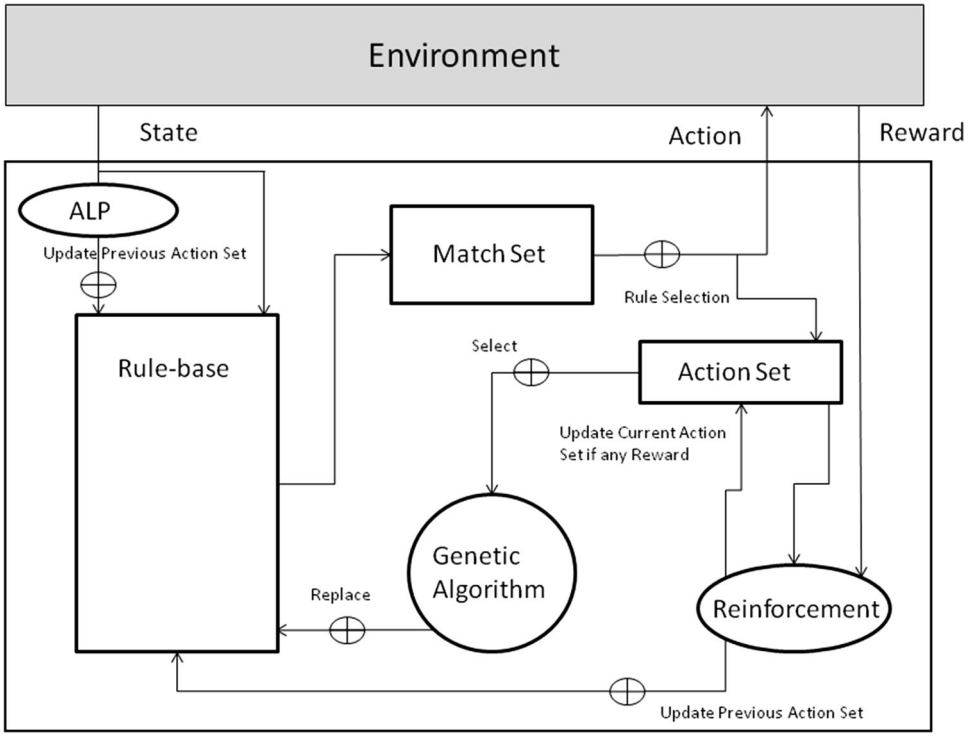

Fig. 9 Schematic of XACS (without XCSF component)

anticipations are least accurate are chosen preferentially over a random action, the ability to learn multiple tasks simultaneously over the same environment by including an XCSF component per task (see [94] for a related study), etc. Given the supervised learning-like nature of building anticipations, they have also been learned in a version of XCSF using a neural network to predict the next state (e.g., [19, 20]).

5 Conclusion

Architecturally, XCS can be traced from ANIMAT via ZCS, with GOFER's triggered niche GA being included. The use of accuracy began with CS-1, although it was focused on the highest reward per niche. Moreover, XCS's generalization pressure shares features with that in CS-1 since it is also based on accuracy and rate of use. In CS-1 predicted rewards are only updated if they are accurate or below the current estimate, with action and GA selection based upon this parameter: more accurate rules are more likely to reproduce. Rule ages are reset after use and deletion is based upon age: more frequently used rules are less likely to be replaced. Thus accurate, more general (frequently used) rules are propagated in CS-1. XCS combines both accuracy, in its pure form, and frequency of use into the selection process of the GA. This creates a generalization pressure but, importantly, also frees the deletion process of the GA to be used to maintain multiple niches in an emergent way thereby addressing one of the main issues in CS-1 that Holland sought to tackle by switching to strength sharing in his subsequent LCS. Much has subsequently been explored with XCS, and there remains much to explore.

XCS has been used effectively to control physical robots in continuous time and space where the action space was discrete and relatively small (e.g., [93]). Wilson [117] has presented a "generalized classifier system" concept whereby LCS can work in a continuous-valued action space. Whilst studies have shown progress in this area for regression problems (e.g., [76]), there is still more to be done for reinforcement learning problems (e.g., [32, 62]).

As highlighted in Bull [14], XCSC can in hindsight be viewed as a type of Artificial Immune System (AIS). For nearly 30 years (starting with [42]) similarities between LCS and AIS have periodically been noted, but the two fields have developed independently. The use of selection within niches of co-active rules is akin to the scheme used in a general class of AIS known as clonal selection algorithms (e.g., CLONALG [36]). The use of a time-delayed evolutionary process is also similar to the dendritic cell AIS (e.g., [50]). It can also be noted that the adaptive affinity threshold finding in XCSC [98] is much like the result reported in Bezerra et al. [4] with an AIS. A potential area for future research would therefore appear to be to explore the cross-fertilization of mechanisms between what are now two relatively mature fields (e.g., see [101] for an overview of AIS).

LCS were presented as an architecture through which to study cognitive systems. Whilst the reinforcement learning element has a clear connection to neuroscience (e.g., [82]), the use of an evolutionary process to build knowledge representations has lacked a strong connection. The general similarities between LCS and artificial neural networks have long been noted (e.g., [86]), and as mentioned above such networks have been used as rules, including spiking models (e.g. [63]). However, there are suggestions that neurogenesis may be significant in adult learning (e.g., [2]) and that such neurons may vary genetically upon production (e.g., [34]). Selectionist models of brains, i.e., forms of neural Darwinism, continue to be developed (e.g., [44]). It has recently been shown [89] that XCSF can be very similar to the locally-weighted projection regression algorithm [108], suggesting its rules may be seen to specify local receptive fields. Another potential area for future research would therefore appear to be to move LCS closer to computational neuroscience.

References

1. Ahluwalia M, Bull L (1999) A genetic programming-based classifier system. In: Banzhaf W et al (eds) GECCO-99: proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Burlington, pp 11–18
2. Becker S (2005) A computational principle for hippocampal learning and neurogenesis. Hippocampus 15(6):722–738
3. Bernado Mansilla E, Garrell J (2003) Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. Evol Comput 11(3):209–238
4. Bezerra G, Barra T, de Castro L, Von Zuben (2005) Adaptive radius immune algorithm for data clustering. In: Pilat C et al (eds) Proceedings of the 4th international conference on artificial immune systems. Springer, New York, pp 290–303
5. Bonelli P, Parodi A, Sen S, Wilson SW (1990) NEWBOOLE: a fast GBML system. In: International conference on machine learning. Morgan Kaufmann, Burlington, pp 153–159
6. Booker L (1982) Intelligent behavior as an adaptation to the task environment. Ph.D. Thesis, the University of Michigan
7. Booker LB (1985) Improving the performance of genetic algorithms in classifier systems. In: Grefenstette JJ (ed) Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum Associates, New York, pp 80–92
8. Booker L (1988) Classifier systems that learn internal world models. Mach Learn 3:161–192
9. Booker L (1989) Triggered rule discovery in classifier systems. In Schaffer J (ed) Proceedings of the international conference on genetic algorithms. Morgan Kaufmann, Burlington, pp 265–274
10. Box G (1957) Evolutionary operation: a method for increasing industrial productivity. J R Stat Soc C 6(2):81–101
11. Bull L (ed) (2004) Applications of learning classifier systems. Springer, New York
12. Bull L (2005) Two simple learning classifier systems. In: Bull L, Kovacs T (eds) Foundations of learning classifier systems. Springer, New York, pp 63–90
13. Bull L (2009) On dynamical genetic programming: simple boolean networks in learning classifier systems. Int J Parallel Emergent Distrib Syst 24(5):421–442
14. Bull L (2011) Towards a mapping of modern AIS and LCS. In: Lio P et al (eds) Proceedings of the tenth international conference on artificial immune systems. Springer, New York pp 371–382
15. Bull L (2014) Exploiting generalisation symmetries in accuracy-based learning classifier systems: an initial study. http://arxiv.org/abs/1401.2949
16. Bull L, Hurst J (2002) ZCS Redux Evol Comput 10(2):185–205
17. Bull L, O'Hara T (2002) Accuracy-based neuro and neuro-fuzzy classifier systems. In: Langdon WB et al (eds) GECCO-2002: proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Burlington, pp 905–911
18. Bull L, Kovacs T (eds) (2005) Foundations of learning classifier systems. Springer, New York
19. Bull L, Lanzi P-L, O'Hara T (2007) Anticipation mappings for learning classifier systems. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, Piscataway, pp 2133–2140
20. Bull L, Studley M, Bagnall A, Whittley I (2007) Learning classifier system ensembles with rule sharing. IEEE Trans Evol Comput 11(4):496–502
21. Bull L, Bernado Mansilla E, Holmes J (eds) (2008) Learning classifier systems in data mining. Springer, New York
22. Butz M (2002) Anticipatory learning classifier systems. Kluwer, New Yokr
23. Butz MV (2006) Rule-based evolutionary online learning systems. Springer, New York
24. Butz MV, Wilson SW (2002) An algorithmic description of XCS. Soft Comput 6(3–4):144–153
25. Butz MV, Goldberg DE (2003) Generalized state values in an anticipatory learning classifier system. In: Butz MV, Sigaud O, Gérard P (eds) Anticipatory behavior in adaptive learning systems. Springer, New York, pp 282–301
26. Butz MV, Kovacs T, Lanzi P-L, Wilson SW (2004) Toward a theory of generalization and learning in XCS. IEEE Trans Evol Comput 8(1):28–46
27. Butz MV, Sastry K, Goldberg DE (2005) Strong, stable, and reliable fitness pressure in XCS due to tournament selection. Genet Program Evol Mach 6(1):53–77
28. Butz MV, Goldberg DE, Lanzi P-L (2005) Gradient descent methods in learning classifier systems: improving XCS performance in multi-step problems. IEEE Trans Evol Comput 9(5):452–473
29. Butz MV, Pelikan M, Llora X, Goldberg DE (2006) Automated global structure extraction for effective local building block processing in XCS. Evol Comput 14(3):345–380
30. Butz MV, Goldberg D, Lanzi P-L, Sastry K (2007) Problem solution sustenance in XCS: markov chain analysis of niche support distributions and the impact on computational complexity. Genet Program Evol Mach 8(1):5–37
31. Butz MV, Lanzi P-L, Wilson SW (2008) Function approximation with XCS: hyperellipsoidal conditions, recursive least squares, and compaction. IEEE Trans Evol Comput 12(3):355–376
32. Casillas J, Carse B, Bull L (2007) Fuzzy XCS: a michigan genetic fuzzy system. IEEE Trans Fuzzy Syst 15(4):536–550
33. Cliff D, Ross S (1995) Adding Temporary Memory to ZCS. Adapt Behav 3(2):101–150
34. Coufal N et al (2009) L1 retrotransposition in human neural progenitor cells. Nature 460:1127–1131
35. Dam H, Abbass H, Lokan C, Yao X (2008) Neural-based learning classifier systems. IEEE Trans Knowl Data Eng 20(1):26–39

36. De Castro L, Von Zuben F (2002) Learning and optimization using the clonal selection principle. IEEE Trans Evol Comput 6(3):239–251
37. Dorigo M, Bersini H (1994) A comparison of Q-learning and classifier systems. In: Cliff D, Husbands P, Meyer J-A, Wilson SW (eds) From animals to animats 3: proceedings of the third international conference on simulation of adaptive behaviour. MIT Press, Cambridge, pp 248–255
38. Drugowitsch J, Barry A (2005) XCS with eligibility traces. In: Beyer HG et al (eds) GECCO-2005: proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Burlington, pp 1851–1858
39. Edakunni N, Brown G, Kovacs T (2011) Online, GA based mixture of experts: a probabilistic model of UCS. In: GECCO-2011: Proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1267–1274
40. Eiben A, Smith J (2003) Introduction to evolutionary computing. Springer, New York
41. Farley B, Clark W (1954) Simulation of self-organizing systems by digital computer. IRE Trans Inf Theory 4:76–84
42. Farmer JD, Packard N, Perelson A (1986) The immune system, adaptation and machine learning. Phys D 22:187–204
43. Fernández A, García S, Luengo J, Bernadó-Mansilla E, Herrera F (2010) Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study. IEEE Trans Evol Comput 14(6):913–941
44. Fernando C, Szathmary E, Husbands P (2012) Selectionist and evolutionary approaches to brain function: a critical appraisal. Front Comput Neurosci 6:24
45. Fraser A (1957) Simulation of genetic systems by automatic digital computers. I. Introduction. Aust J Biol Sci 10:484–491
46. Frey P, Slate D (1991) Letter recognition using holland-style adaptive classifiers. Mach Learn 6:161–182
47. Gererd P, Sigaud O (2001) YACS: combining dynamic programming with generalization in classifier systems. In: Lanzi P-L, Stolzmann W, Wilson SW (eds) Advances in learning classifier systems: proceedings of the third international workshop on learning classifier systems. Springer, New York, pp 52–69
48. Giani A, Baiardi F, Starita A (1995) PANIC: a parallel evolutionary rule based system. In: McDonnell J, Reynolds R, Fogel D (eds) Proceedings of the fourth annual conference on evolutionary programming. MIT Press, Cambridge, pp 753–772
49. Goldberg D (1985) Genetic algorithms and rule learning in dynamic system control. In: Grefenstette JJ (ed) Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum Associates, New York, pp 8–15
50. Greensmith J, Feyereisl J, Aickelin U (2008) DCA: some comparison. Evol Intel 1(2):85–112
51. Hartley A (1999) Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments. In: Banzhaf W et al (eds) GECCO-99: proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Burlington, pp 266–273
52. Hercog L, Fogarty TC (2002) Coevolutionary classifier systems for multi-agent simulation. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, Piscataway, pp 1798–1803
53. Hoffmann J (1993) Vorhersage und Erkenntnis. Hogrefe, Goettingen
54. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
55. Holland JH (1976) Adaptation. In: Rosen and Snell (eds) Progress in theoretical biology, vol 4. Plenum, Berlin, pp 263–293
56. Holland JH (1980) Adaptive algorithms for discovering and using general patterns in growing knowledge bases. Int J Policy Anal Inf Syst 4(3):245–268
57. Holland JH (1985) Properties of the bucket brigade. In: Grefenstette JJ (ed) Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum Associates, New York, pp 1–7
58. Holland JH (1986) Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: Michalski R, Carbonell J, Mitchell T (eds) Machine learning, an artificial intelligence approach. Morgan Kaufmann, Burlington, pp 593–623
59. Holland JH (1990) Concerning the emergence of tag-mediated lookahead in classifier systems. Phys D 42:188–201
60. Holland JH, Reitman JH (1978) Cognitive systems based in adaptive algorithms. In Waterman DA, Hayes-Roth F (eds) Pattern-directed inference systems. Academic Press, New York, pp 313–329
61. Holland JH, Holyoak KJ, Nisbett RE, Thagard PR (1986) Induction: processes of inference, learning and discovery. MIT Press, Cambridge
62. Howard D, Bull L, Lanzi P-L (2009) Continuous actions in continuous space and time using self-adaptive constructivism in neural XCSF. In: GECCO-2009: proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1219–1226
63. Howard D, Bull L, Lanzi P-L (2010) A spiking neural representation for XCSF. In: Proceedings of the IEEE congress on evolutionary computation. IEEE Press, Piscataway, pp 1–8
64. Hurst J, Bull L (2002) A self-adaptive XCS. In: Lanzi P-L, Stolzmann W, Wilson SW (eds) Advances in learning classifier systems: proceedings of the 4th international workshop on learning classifier systems. Springer, New York, pp 57–73
65. Iqbal M, Browne W, Zhang M (2013) Evolving optimum populations with XCS classifier systems—XCS with code fragmented action. Soft Comput 17(3):503–518
66. Iqbal M, Browne W, Zhang M (2014) Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems. IEEE Trans Evol Comput 18(4):465–480
67. Kovacs T (1997) XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In: Roy, Chawdhry, Pant (eds) Soft computing in engineering design and manufacturing. Springer, New York, pp 59–68
68. Lanzi P-L (2008) Learning classifier systems: then and now. Evol Intel 1(1):63–82
69. Lanzi P-L, Perrucci A (1999) Extending the representation of classifier conditions part II: from messy coding to S-expressions. In: Banzhaf W et al (eds) GECCO-99: proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Burlington, pp 345–352
70. Lanzi P-L, Riolo R (2000) A roadmap to the last decade of learning classifier system research. In: Lanzi P-L, Stolzmann W, Wilson SW (eds) Learning classifier systems: from foundations to applications. Springer, New York, pp 33–62
71. Lanzi P-L, Wilson SW (2000) Toward optimal classifier system performance in non-markov environments. Evol Comput 8(4):393–418
72. Lanzi P-L, Loiacono D, Wilson SW, Goldberg D (2007) Generalization in the XCSF classifier system: analysis, improvement, and extension. Evol Comput 15(2):133–168
73. Loiacono D, Lanzi P-L (2009) Recursive least squares and quadratic prediction in continuous multistep problems. In: Bacardit J et al (eds) Learning classifier systems: revised selected papers. Springer, New York, pp 70–86
74. Orriols-Puig A, Bernado Mansilla E (2008) Revisiting UCS: description, fitness sharing, and comparison with XCS. In: Bacardit J et al (eds) Learning classifier systems: revised selected papers. Springer, New York, pp 96–116

75. Orriols-Puig A, Casillas J, Bernadó Mansilla E (2009) Fuzzy-UCS: a michigan-style learning fuzzy-classifier system for supervised learning. IEEE Trans Evol Comput 13(2):260–283
76. Preen R, Bull L (2013) Dynamical genetic programming in XCSF. Evol Comput 21(3):361–388
77. Qian L, Shi Y, Gao Y, Yin H (2013) Voting-XCSc: a consensus clustering method via learning classifier system. In: Yin H et al (eds) Intelligent data engineering and automated learning—IDEAL. Springer, New York, pp 603–610
78. Riolo R (1991) Lookahead planning and latent learning in a classifier system. In: Meyer J-A, Wilson SW (eds) From animals to animats: proceedings of the first international conference on simulation of adaptive behaviour. MIT Press, Cambridge, pp 316–326
79. Roberts G (1993) Dynamic planning for classifier systems. In: Forrest S (ed) Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann, Burlington, pp 231–237
80. Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3:211–229
81. Samuel AL (1967) Some studies in machine learning using the game of checkers. II. Recent progress. IBM J Res Dev 11:601–617
82. Schultz W (1998) Predictive reward signal of dopamine neurons. J Neurophysiol 68:1190–1208
83. Seward J (1949) An experimental analysis of latent learning. J Exp Psychol 39:177–186
84. Shannon C (1950) Programming a computer for playing chess. Phil Mag 41:256–275
85. Smith SF (1980) A learning system based on genetic adaptive algorithms. PhD Thesis, University of Pittsburgh
86. Smith R, Cribbs H (1994) Is a learning classifier system a type of neural network? Evol Comput 2(1):19–36
87. Smith R, Jiang M, Bacardit J, Stout M, Krasnogor N, Hirst J (2010) A learning classifier system with mutual-information-based fitness. Evol Intel 3(1):31–50
88. Stalph P, Llorà X, Goldberg D, Butz MV (2012) Resource management and scalability of the XCSF learning classifier system. Theoret Comput Sci 425:126–141
89. Stalph P, Rubinsztajin J, Sigaud O, Butz MV (2012) Function approximation with LWPR and XCSF: a comparative study. Evol Intel 5(2):103–116
90. Stolzmann W (1998) Anticipatory classifier systems. In: Koza et al (eds) Genetic programming 1998: proceedings of the third annual conference. Morgan Kaufmann, Burlington, pp 658–654
91. Stone C, Bull L (2003) For Real! XCS with continuous-valued inputs. Evol Comput 11(3):299–336
92. Stone C, Bull L (2005) Comparing XCS and ZCS on noisy continuous-valued environments. Technical report: UWELCSG05-002. http://www.cems.uwe.ac.uk/lcsg
93. Studley M, Bull L (2005) X-TCS: accuracy-based learning classifier system robotics. In: Proceedings of the IEEE congress on evolutionary computation. IEEE, pp 2099–2106
94. Studley M, Bull L (2006) Using the XCS classifier system for multi-objective reinforcement learning problems. Artif Life 13(1):69–86
95. Sutton R, Barto A (1981) Toward a modern theory of adaptive networks: expectation and prediction. Psychol Rev 88:135–170
96. Sutton R, Barto A (1998) Reinforcement learning. MIT Press, Cambridge
97. Tammee K, Bull L, Ouen P (2006) A learning classifier system approach to clustering. In: Proceedings of the 6th international conference on intelligent systems design and applications. IEEE, pp 621–626
98. Tammee K, Bull L, Ouen P (2007) Towards clustering with XCS. In: Thierens D et al (eds) GECOO-2007: proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1854–1860
99. Thorndike E (1911) Animal intelligence. Macmillan Company, New York
100. Tibshirani R, Walther G, Hastie T (2000) Estimating the number of clusters in a dataset via the gap statistic. J R Stat Soc B 63:411–423
101. Timmis J, Andrews P, Owens N, Clark E (2008) An interdisciplinary perspective on artificial immune systems. Evol Intel 1(1):5–26
102. Tomlinson A, Bull L (2002) An accuracy-based corporate classifier system. Soft Comput 6(3–4):200–215
103. Tran T, Sanza C, Duthen Y, Nguyen D (2007) XCSF with computed continuous action. In: Thierens D et al (eds) GECCO-07: proceedings of the genetic and evolutionary computation conference. ACM Press, New York, pp 1861–1868
104. Turing A (1948) Intelligent machinery. Reprinted in: Copeland J. (2004) The essential turing. Oxford University Press, Oxford, pp 395–432
105. Urbanowicz R, Moore J (2009) Learning classifier systems: a complete introduction, review and roadmap. J Artif Evol Appl 1:1–25
106. Valenzuela-Rendón M (1991) The fuzzy classifier system: a classifier system for continuously varying variables. In: Belew R, Booker L (eds) Proceedings of the 4th international conference on genetic algorithms. Morgan Kaufmann, Burlington, pp 346–353
107. Venturini G (1994) Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique. Thèse de Docteur en Science (Informatique), Université de Paris-Sud
108. Vijayakumar S, D'Souza A, Schall S (2005) Incremental on-line learning in high dimensions. Neural Comput 17(12):2602–2634
109. Watkins CJ (1989) Learning from delayed rewards. Ph.D. Thesis, Cambridge University
110. Wilson SW (1985) Knowledge growth in an artificial animal. In: Grefenstette JJ (ed) Proceedings of the first international conference on genetic algorithms and their applications. Lawrence Erlbaum Associates, New York, pp 16–23
111. Wilson SW (1987) Classifier systems and the animat problem. Mach Learn 2:219–228
112. Wilson SW (1994) ZCS: a zeroth-level classifier system. Evol Comput 2(1):1–18
113. Wilson SW (1995) Classifier fitness based on accuracy. Evol Comput 3(2):149–176
114. Wilson SW (1998) Generalization in the XCS classifier system. In: Koza et al (eds) Genetic programming 1998: proceedings of the 3rd annual conference. Morgan Kaufmann, Burlington, pp 322–334
115. Wilson SW (2001) Function approximation with a classifier system. In: Spector L et al (eds) GECCO-01: proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, Burlington, pp 974–981
116. Wilson SW (2002) Classifiers that approximate functions. Nat Comput 1(1):211–233
117. Wilson SW (2007) Three architectures for continuous action. In: Bacardit J et al (eds) Learning classifier systems: revised selected papers. Springer, New York, pp 239–257
118. Wilson SW (2008) Classifier conditions using gene expression programming. In Bacardit J et al (eds) Learning classifier systems: revised selected papers. Springer, New York, pp 206–217
119. Wilson SW, Goldberg DE (1989) A critical review of classifier systems. In: Schaffer J (ed) Proceedings of the 3rd international conference on genetic algorithms. Morgan Kauffman, San Francisco, pp 244–255
120. Wyatt D, Bull L (2004) A memetic learning classifier system for describing continuous-valued problem spaces. In: Krasnagor N, Hart W, Smith J (eds) Recent advances in memetic algorithms. Springer, New York, pp 355–396
121. Xu R, Wunsch D (2009) Clustering. IEEE Press, Piscataway