47. Learning Classifier Systems

Martin V. Butz

Learning Classifier Systems (LCSs) essentially combine fast approximation techniques with evolutionary optimization techniques. Despite their somewhat misleading name, LCSs are not only systems suitable for classification problems, but may be rather viewed as a very general, distributed optimization technique. Essentially, LCSs have very high potential to be applied in any problem domain that is best solved or approximated by means of a distributed set of local approximations, or predictions. The evolutionary component is designed to optimize a partitioning of the problem domain for generating maximally useful predictions within each subspace of the partitioning. The predictions are generated and adapted by the approximation technique. Generally any form of spatial partitioning and prediction are possible – such as a Gaussian-based partitioning combined with linear approximations, yielding a Gaussian mixture of linear predictions. In fact, such a solution is developed and optimized by XCSF (XCS for function approximation). The LCSs XCS (X classifier system) and the function approximation version XCSF, indeed, are probably the most well-known LCS architectures to date. Their optimization technique is very-well balanced with the approximation technique: as long as the approximation technique yields reasonably good solutions and evaluations of these solutions fast, the evolutionary component will pick-up on the evaluation signal and optimize the partitioning. This chapter provides historical background on LCSs. Then XCS and XCSF are introduced in detail providing enough information to be able to implement, understand, and apply these systems. Further LCS architectures are surveyed and their potential for future research and for applications is discussed. The conclusions provide an outlook on the many possible future LCS applications and developments.

Learning classifier systems (LCSs) are machine learning algorithms that combine gradient-based approximation with evolutionary optimization. Due to this flexibility, LCSs have been successfully applied to classification and data mining problems, reinforcement learning (RL) problems, regression problems, cognitive map learning, and even robot control problems. The main feature of LCSs is their innovative combination of two learning principles; whereas gradient-based approximation adapts local, predictive approximations of target function values, evolutionary optimization structures individual classifiers to enable the formation of effectively distributed and accurate approximations. The two learning methods interact bidirectionally in

that the gradient-based approximations yield local fitness quality estimates of the generated approximations, which the evolutionary optimization technique uses for optimizing classifier structures. Concurrently, the evolutionary optimization technique is generating new classifier structures, which again need to be evaluated by the gradient-based approach in competition with the other, locally overlapping, interacting classifiers.

Due to the innovative combination of two learning and optimization techniques, LCSs are often perceived as being hard to understand. Facet-wise analyses of the individual LCS components and their interactions, however, give both mathematical scalability bounds for learning and an intuitive understanding of the systems in general. Moreover, the currently most common LCS, which is the XCS classifier system (note that the *X* in XCS does not really encode any particular acronym according to the system creator Wilson), is comparatively easy to understand, to tune, and to apply. Thus, the core of this chapter focuses on XCS, gives a facet-wise overview of its functionality, details several enhancements, and highlights various successful application domains. However, XCS is also compared with other LCS architectures and LCSs in general are compared with other machine learning techniques.

This chapter starts with a historical perspective, providing information on the beginnings of LCSs and establishing some terminology background. We then introduce the XCS classifier system providing a detailed system overview as well as theoretical and facet-wise conceptual insights on its performance. Also tricks and tweaks are discussed to tune the system to the problem at hand. Next, the XCS counterpart for regression problems, XCSF, is introduced. Focusing then on the application-side, LCS applications to data mining tasks and to behavioral learning and cognitive modeling tasks are surveyed. We cover various LCS architectures that have been successfully applied in the data mining realm. With respect to behavioral learning, we point out the relation of LCSs to reinforcement learning. Moreover, we cover anticipatory learning classifier systems (ALCSs) – which learn predictive schema models of the environment rather than reward prediction maps – and we introduce the modified XCSF version that can effectively learn a redundant forward-inverse kinematics model of a robot arm. A summary and conclusions wrap up the chapter.

47.1 Background

Learning classifier systems (LCS) were proposed over 30 years ago by *Holland* [47.1–3]. Originally, *Holland* and *Reitman* actually called LCSs *cognitive systems* [47.4], focusing on problems related to reinforcement learning (RL) [47.5, 6]. His cognitive system developed a *memory* of *classifiers*, where each classifier consisted of a condition part (*taxon*), an action part (originally consisting of a *message*, and an *effector bit*), a payoff prediction part, and several other parameters that stored the age, the application frequency, and the *attenuation* of the classifier.

Concurrently with the development of temporal difference learning techniques in RL – such as the now well-known state-action-reward-state-action (SARSA) algorithm [47.6] – *Holland* and *Reitman* introduced the *bucket brigade* algorithm [47.4, 7], which also distributes reward backwards in time with a discounting mechanism. In addition, the *attenuation* parameter in a classifier realized something similar to an *eligibility trace* in RL – distributing a currently encountered reward also to classifiers that were active several time steps ago and that thus indirectly led to gaining the currently experienced reward. Meanwhile, Holland's cognitive system applied a genetic algorithm (GA) [47.1, 8] as its second learning mechanism. The GA modified the *taxa* in Holland and Reitman's cognitive system.

In sum, the first actual LCS implementation, i. e., the *cognitive system* by *Holland* and *Reitman* [47.4], was ahead of its time. It implemented various reward-related ideas that were later established in the reinforcement learning community – and can now partially be regarded as standard RL techniques. However, the combination with GAs yielded a highly interactive and very complex system that was and still is hard to analyze. Thus, while proposing a highly innovative cognitive learning approach, the applicability of the system remained limited at the time.

47.1.1 Early Applications

Nonetheless, early applications of LCSs were published in the 1980s. *Smith* developed a poker decision making system [47.9] based on *De Jong*'s approach to LCSs [47.10]. *Booker* worked on animal-like

automation based on the cognitive systems architecture [47.11]. *Wilson* proposed and worked on the *animat problem* with LCS architectures derived from Holland and Reitman's cognitive systems approach [47.12, 13]. *Goldberg* solved a gas pipeline control task with a simplified version of the cognitive system architecture [47.8, 14]. Despite these successful early applications, a decade passed until a growing research community developed that worked on learning classifier systems.

47.1.2 The Pitt and Michigan Approach

Two fundamentally different LCS approaches were pursued from early on. The Pitt approach was fostered by the work of *De Jong* et al. [47.10, 15, 16]. On the other hand, the Michigan approach developed in the further years at Michigan under the supervision of *John H. Holland* [47.11, 14, 17, 18]. Diverse perspectives on the Michigan approach can be found in [47.19].

The essential difference between the two approaches is that in the Pitt approach rule sets are evolved where each particular rule set constitutes an individual for the GA. In contrast, in the Michigan approach one set of rules is evolved and each rule is an individual for the GA. As a consequence, the Pitt-style LCSs are much closer to general GAs because each individual constitutes an overall problem solution. In the Michigan-style LCSs, on the other hand, each individual only applies in a subspace of the overall problem and only the whole set of rules that evolves constitutes the overall problem solution. Figure 47.1 illustrates this fundamental contrast between the two approaches.

As a consequence of this contrast, Pitt-style LCSs usually apply rather standard GA approaches. The whole population of rule sets is evolved. For fitness evaluation purposes, each set of rules needs to be evaluated in the problem environment addressed. On the other hand, Michigan-style LCSs need to continuously interact with an environment to sufficiently evaluate all the rules in the rule set – essentially exploring all the environmental subspaces to make sure all rules can develop a sufficiently useful fitness estimate. This continuous interaction and the typical interacting components of Michigan-style LCSs are illustrated in further detail in Fig. 47.2. Due to the continuously developing fitness estimates, often a more steady-state, niched GA is applied online in Michigan-style LCSs. The undertaken updates then depend directly on the current interaction and thus on the current subset of rules relevant in the experienced interaction. The steady-state, niched GA optimizes the internal knowledge base iteratively depending on the incoming learning samples.

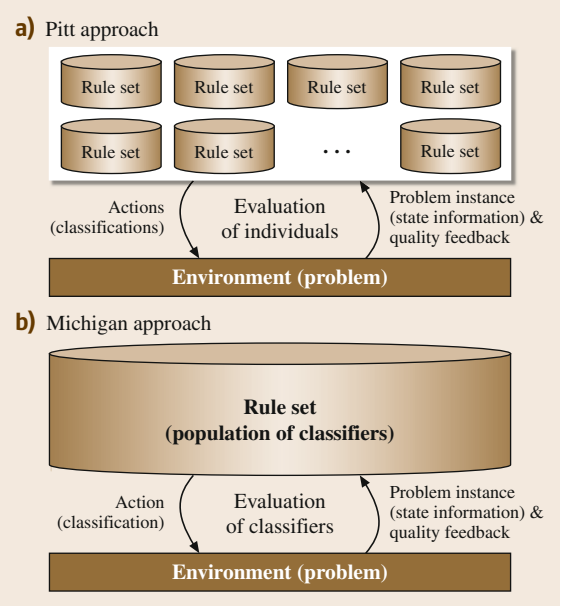


Fig. 47.1a,b While the Pitt approach to LCSs evolves a population of sets of rules, in the Michigan approach there is only one set of rules (i. e., the population) that is evolved

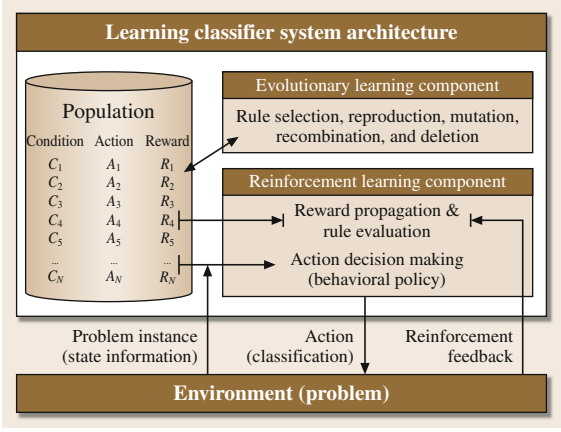


Fig. 47.2 LCSs consist of a knowledge base (population of classifiers), a genetic algorithm for rule structure evolution, and a reinforcement learning component for rule evaluation, reward propagation, and decision making. The system interacts with its environment or problem iteratively learning online

In summary, Pitt-style LCSs evaluate and optimize their rule sets globally based on sets of problem instances. They usually learn *offline*. Michigan-style LCSs evaluate and optimize their set of rules online while interacting with the problem, iteratively perceiving problem instances. The major qualities of Pitt-style LCSs are that they evolve competing global problem solutions in the form of sets of rules. Evolutionary rule structure optimization is used – typically evolving small sets of rules (10 s). Michigan-style LCSs, on the other hand, are designed to develop one distributed, locally optimized problem solution by combining local gradient-based approximation techniques with steady-state, niched GAs. In consequence, typically larger, more distributed sets of rules develop yielding problem solutions with potentially 1000 s of rules.

47.1.3 Basic Knowledge Representation

Because an exemplary knowledge representation was already discussed for the early *cognitive system* implementation of [47.4], we now provide a general sketch of the knowledge representation typically found in Michigan-style LCSs.

The knowledge representation of an LCS consists of a finite *population of classifiers* (that is, a finite set of rules). This population of classifiers essentially represents the current knowledge of the LCS about the problem the system is applied to. Each rule – or *classifier* – usually consists of a *condition* and an *action part*, as well as a *prediction* and a *fitness* estimate. The condition part specifies the problem subspace in which the classifier is applicable. When the condition part is satisfied given a particular problem instance, a classifier is said to *match* that problem instance. The action part specifies an action that may be executed, or a classification that may be tested. The prediction specifies the expected reward, or feedback value, given the specified action was executed under the specified contextual conditions. The fitness estimates the value of this classifier relative to other, competing classifiers. In the early approaches, fitness was often simply equal to the prediction value. In the currently established LCSs, fitness typically estimates the accuracy of the prediction.

Michigan-style LCSs usually learn online about a problem, iteratively perceiving or actively generating problem instances. Given a particular problem instance, first, the system forms a *match set* of those classifiers in the population whose conditions match. Next, the system decides on an action or classification and executes it. Classifiers in the match set that specify the executed action constitute the current *action set*. After feedback is received, the predictions of the classifiers in the action set are adjusted. From the classifier prediction estimates, a fitness estimate is derived for each classifier. Finally, the steady-state GA is applied to the match set or the population as a whole. The GA modifies classifier structures by reproducing, mutating, and recombining well-performing classifiers and by deleting ill-performing ones. In contrast to the Michigan approach, Pitt-style LCSs evaluate their sets of rules typically independently of each other in the provided problem. The GA exchanges rules and rule-structures within and across the sets of rules.

A Michigan-style LCS consequently is an interactive, online learning system. It maintains a population of classifiers as its knowledge base. It applies a niched, steady-state genetic algorithm for gradual rule structure evolution; it applies a gradient-based learning component for rule evaluation – yielding prediction and fitness estimates. Michigan-style LCSs are often applied in RL scenarios in which reward estimates need to be propagated and action decisions are made based on the learned reward prediction estimates. In this case, typically techniques similar to SARSA learning or Q-learning are applied. Figure 47.2 shows the basic components of a Michigan-style LCS as well as their interactions.

The earliest Michigan-style LCS implementation is the introduced cognitive system CS1 [47.4]. After various early applications of LCSs, *Wilson* set a milestone in LCS research by introducing the zeroth level classifier system ZCS [47.20] and the now most prominent and well-known LCS: the XCS classifier system [47.21]. Both systems were explicitly compared to the very well-known Q-learning [47.22] technique from the RL community, offering with ZCS and XCS two learning classifier systems that can learn Q-value functions with a compact highly generalized rule-based representation.

In the following, we now first give a precise introduction to the XCS classifier system. We then also introduce the real-valued version for solving regression problems, with a Gaussian mixture of linear approximations, i. e., XCSF. After that, we provide spot-lights on various current application domains where various types of LCSs, including XCS(F), have produced highly competitive problem solutions, when compared to other machine learning techniques and regression algorithms.

47.2 XCS

Wilson introduced the XCS classifier system in 1995 [47.21]. The two main novel features of XCS in comparison to earlier Michigan-style LCSs are its accuracy-based fitness estimation and its niche-based application of the evolutionary component. The introduction of accuracy-based fitness essentially decoupled the classifier fitness estimate from the reward prediction, enforcing that XCS learned *complete* payoff landscapes rather than only estimates for those subspaces where high reward is encountered. In addition, *Wilson* related XCS directly to Q-Learning [47.21, 22]. Much later, even a relation to Kalman filtering and general regression tasks was made mathematically explicit [47.23, 24]. The niche-based GA reproduction combined with population-wide deletion enabled a much more focused GA-based optimization of classifier structures as well as the generalization of classifier structures based on the sampling distribution [47.25]. In consequence, XCS is an LCS that is designed to evolve not only the best solution to a problem, but it evolves all alternative solutions with associated Q-value estimations and variance estimations of the respective Q-value estimates. Due to its GA design and fitness definition, XCS strives to approximate the full Q-table of a problem with a maximally accurate and maximally compact classifier-based representation.

Despite its original strong relation to Q-learning and RL in general, XCS has also been applied successfully to classification problems and regression problems. In the former case, XCS identifies locally relevant features for the generation of maximally accurate classification estimates. In the latter case, XCS optimizes the distribution and structure of local, typically linear estimators for a maximally accurate approximation of the function surface. Thus, despite its original strong relation to RL, XCS is a much more generally applicable learning system that can solve single-step classification or regression problems as well as multi-step RL problems, which are typically defined as Markov decision processes.

47.2.1 System Overview

XCS evolves one population of classifiers. Classifier structures are optimized by means of a steady-state GA. A classifier consists of a condition part C, an action part A, reward prediction r, reward prediction error ε, and fitness f estimates. While the condition and action structures are iteratively optimized by the steady-state GA, the estimates are adjusted using the Widrow–Hoff delta rule [47.26] based on an approximation of the Q-value signal.

While condition and action parts can be generally represented in any way desired [47.25], in this overview we focus on binary problems and a ternary representation of the condition part. Conventionally, the condition part C is coded by $C \in \{0, 1, \#\}^L$, where the # symbol matches zero and one. Condition C essentially specifies a hypercube within which the classifier *matches* and can be said to cover a certain *volume* of the complete problem space. Action part $A \in \mathcal{A}$ defines an action or classification from a provided finite set of possible actions $\mathcal{A}$. Reward prediction $r \in \mathbb{R}$ estimates the moving average of the received reward in the recent activations of the classifier. Reward prediction error ε estimates the moving average of the absolute error of the reward prediction. Finally, fitness $f \in [0, 1]$ estimates the moving average of the relative accuracy of the classifier compared to the competing classifiers in the activated match sets (or action sets). The larger the fitness estimate, the on average larger the accuracy of a classifier in comparison to all classifiers that encode the same action and whose condition parts define overlapping subspaces.

Each classifier also maintains several additional parameters. The *action set size estimate as* estimates the moving average of the action sets the classifier was part of. It is updated similarly to the reward prediction r. A *time stamp ts* specifies the last time the classifier was part of a GA competition. An *experience* counter *exp* specifies the number of applied parameter updates. The numerosity *num* specifies the number of (micro-) classifiers, this macro-classifier actually represents – mainly for saving computation time.

Learning usually starts with an empty population. The problem faced is sampled iteratively, encountering particular problem instances $s \in S$. The set of all matching classifiers in the classifier population $[P]$ is termed the *match set* $[M]$. If some action in $\mathcal{A}$ is not represented in $[M]$, a *covering mechanism* is applied. Covering creates classifiers that match s (inserting #-symbols in the new C with a probability $P_\#$ at each position) and that specify the unrepresented actions. $[M]$ essentially contains all the knowledge of XCS about the current problem instance. Given $[M]$, XCS estimates the payoff for each possible action forming a *prediction ar-*

ray $P(\mathcal{A})$,

$$P(A) = \frac{\sum_{\mathrm{cl}.A=A \wedge \mathrm{cl} \in [M]} \mathrm{cl}.r \cdot \mathrm{cl}.f}{\sum_{\mathrm{cl}.A=A \wedge \mathrm{cl} \in [M]} \mathrm{cl}.f} , \tag{47.1}$$

where classifier parameters are addressed using the dot notation. $P(A)$ computes the fitness-averaged Q-value estimates for each action in the current state s. Thus, $P(A)$ can be used to decide on the currently most promising action.

Any action selection policy may be applied, such as choosing the action with the largest Q-value expectation. Because XCS relies on exploring the complete problem spaces, however, it is important that all actions are applied sufficiently frequently. Alternatively, also the prediction error estimates may be considered for action selection – choosing, for example, that action with the highest fitness-averaged ε value with the aim of maximizing information gain (see also more elaborate techniques surveyed recently in the computational intelligence literature [47.27]).

After the choice of an action A, an *action set* $[A]$ is formed, which contains all classifiers in $[M]$ that specify the chosen action. Moreover, the chosen action is executed, feedback is received in the form of scalar reward $R \in \mathbb{R}$, and the next problem instance may be perceived. In conjunction with the maximum $P(A)$ derived from the resulting match set, the $[A]$ formed is updated according to the estimated Q-value signal, which is $R + \gamma \max_{A \in \mathcal{A}} P(A)$. Moreover, the steady-state GA may be applied, reproducing two classifiers in $[A]$, but choosing classifiers from $[P]$ for deletion. In classification problems – often also termed *single–step problems* – the Q-learning update only considers the immediate reward R. Figure 47.3 illustrates the iterative learning process applied in XCS.

Rule Evaluation

To evaluate the classifiers, it is crucial to update their parameter estimates and derive a relative fitness estimate. Parameter updates are applied iteratively in respective action sets. Usually, the prediction error is updated before the prediction and the fitness. Other parameters may be updated in any order.

In particular, the reward prediction error ε of each classifier in $[A]$ is updated by

$$\varepsilon \leftarrow \varepsilon + \beta(|\rho - R| - \varepsilon) , \tag{47.2}$$

where $\rho = R$ in classification problems and

$$\rho = R + \gamma \max_{A \in \mathcal{A}} P(A)$$

in multi-step reinforcement learning problems. Parameter $\beta \in [0, 1]$ specifies a learning rate, which is typically set to values between 0.05 and 0.2. The higher the value of β is, the more the ε value depends on the most recent problem interactions. Next, the reward prediction r of each classifier in $[A]$ is updated by

$$r \leftarrow r + \beta(\rho - r) . \tag{47.3}$$

Note that XCS essentially applies Q-learning updates, where Q-values are not approximated by a tabular entry but by a collection of rules expressed in the prediction array $P(\mathcal{A})$ [47.21].

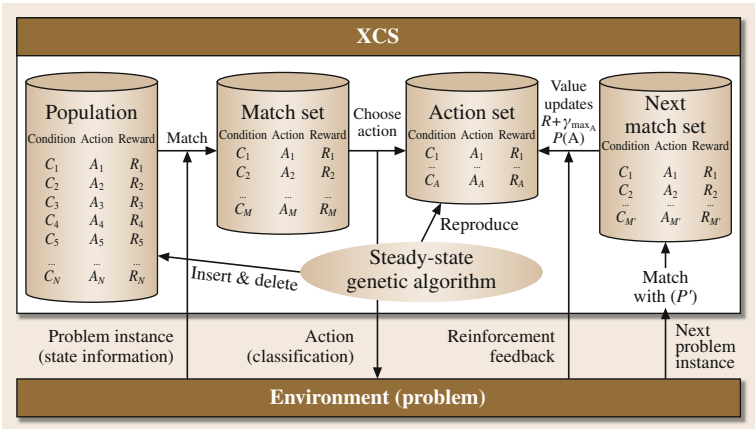


Fig. 47.3 The XCS classifier system learns iteratively online. With each iteration it forms a match set given the current problem instance. Next, it chooses an action or classification and applies it. After the perception of feedback, the classifiers in the corresponding action set $[A]$ are updated and the steady-state GA is applied. After that, the next problem iteration proceeds

To update the fitness estimate of each classifier in $[A]$, a current scaled relative accuracy κ' is determined.

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha \left(\frac{\varepsilon_0}{\varepsilon}\right)^{\nu} & \text{otherwise} \end{cases} , \tag{47.4}$$

$$\kappa' = \frac{\kappa \cdot \text{num}}{\sum\limits_{\text{cl} \in [A]} \text{cl}.\kappa \cdot \text{cl.num}} . \tag{47.5}$$

κ essentially measures the current inverse error of a classifier. ε_0 specifies the targeted error below which a classifier is considered maximally accurate. κ' then determines the current relative accuracy with respect to all other classifiers in the current action set $[A]$. Thus, each classifier in $[A]$ competes for a limited fitness resource, which is distributed relative to the current accuracy estimates κ. Finally, the fitness estimate f is updated given the current κ' by

$$f \leftarrow f + \beta(\kappa' - f) . \tag{47.6}$$

In effect, fitness reflects the moving average, set-relative accuracy of a classifier. As before, β controls the sensitivity of the fitness estimates to changes in the population.

The action set size estimate *as* is updated similarly to the reward prediction R but with respect to the current action set size $|[A]|$

$$as \leftarrow as + \beta(|[A]| - as) , \tag{47.7}$$

resulting in an action set size adaptation to changes $|[A]|$ in an order similar to the fitness changes. Parameters r, ε, and *as* are updated using the *moyenne adaptive modifiée* technique [47.28]. This technique sets parameter values directly to the average of the so far encountered cases until the resulting update is smaller than β (which is the case after $1/\beta$ updates). Finally, the experience counter *exp* is increased by one. If the GA is applied, the time stamps *ts* of all classifiers in $[A]$ are set to the current iteration time t.

Rule Evolution

XCS applies a steady-state genetic algorithm (GA) for rule evolution. Given a current action set $[A]$, the GA is invoked if the average time since the last GA application (stored in parameter *ts*) in $[A]$ is larger than threshold θ_{GA}. This mechanism is applied to ensure sufficient evaluation of classifiers, as well as to control unbalanced sampling. The higher the threshold θ_{GA} is, the slower evolution proceeds, but also the less prone XCS is to unbalanced problem sampling [47.29].

The steady-state GA first selects two parental classifiers for reproduction in $[A]$. While this selection process was done by proportionate selection based on fitness in the original XCS, more recently it was shown that tournament selection can improve the robustness of the system highly significantly [47.30]. Tournament selection in XCS chooses the classifier with the highest fitness from a tournament of randomly chosen classifiers from $[A]$. The tournament size is usually set relative to the current action set size $|[A]|$ to $\tau \cdot |[A]|$. Two classifiers are selected in two independent tournaments. The selected classifiers are reproduced generating the offspring. Crossover and mutation are applied to the offspring. The parents stay in the population. Mutation usually changes each condition and action symbol randomly with a certain probability μ. Crossover exchanges condition and action symbols. Often, simple uniform crossover is applied (exchanging each symbol with a probability of 0.5). However, also more sophisticated estimation of distribution (EDAs) algorithms have been applied for more effective building block processing [47.31].

The offspring parameters are initialized by setting prediction R, ε, f, and *as* to the parental values. Fitness f is often decreased to 10% of the parental fitness. Experience counter exp and numerosity num are set to one.

The resulting offspring classifiers are finally added to the population. In this case, *GA subsumption* may be applied [47.32] to stress generalization. *GA subsumption* searches for another classifier in $[A]$ that may subsume an offspring classifier. This classifier must have a more general condition than the offspring classifier, its error estimate must be below ε_0, and its experience counter must be sufficiently high ($\text{exp} > \theta_{\text{sub}}$). If such a classifier is found, the offspring is *subsumed*, increasing the numerosity of the more general classifier by one and discarding the offspring.

The population of classifiers $[P]$ is maximally of finite size N. When this size is exceeded after offspring insertion, classifiers are deleted from $[P]$. Fitness proportionate selection is applied depending on the action set size estimates *as*. Note that tournament selection is not suitable in this case because a balance in the action set sizes is most desirable. The likelihood of deletion of a classifier is further increased by a factor $\bar{f}/f$ if this classifier is *experienced* $\text{exp} > \theta_{\text{del}}$ and additionally if its fitness f is below a fraction δ of the average fitness $\bar{f}$ in the population.

47.2.2 When and How XCS Works

From the description above it may seem hard to understand why XCS learns successfully. This section provides intuition about when and how XCS works and points to relevant literature that quantifies the sketched-out intuition.

The two interacting learning components, which are gradient-based rule evaluation and evolutionary-based rule evolution, are strongly interactive. From an evolutionary point of view, several evolutionary pressures yield particular learning biases. Since reproduction is designed to maximize fitness, XCS strives to develop maximally accurate classifiers applying a *fitness pressure* [47.33]. Meanwhile, rules are selected in [*A*] for reproduction but they are selected in [*P*] for deletion. Since the classifier conditions in [*A*] will on average cover a larger subspace, i. e., they have a larger *volume* than the average condition volumes of classifiers in [*P*], more general classifiers will be reproduced on average (when ignoring the fitness pressure for the moment), yielding a sampling-dependent *generalization pressure* [47.33]. In consequence, it has been put forward that XCS strives to evolve a *complete problem solution* that is represented by *maximally general* classifiers that are meanwhile *maximally accurate* (error below the threshold ε_0). The resulting problem solution representation was previously termed the *optimal solution representation* [*O*] [47.34].

While these evolutionary pressures generally describe how the GA in XCS works, successful rule evolution still relies on sufficiently accurate fitness signals. Thus, rule evaluation needs to have enough time to estimate rule fitness before expected rule deletion. This leads to a *covering bound*, which quantifies the need for a sufficiently large population size given a particular initial condition volume. Moreover, each particular problem can be assumed to have a certain complexity in terms of subspace sizes that need to be separated for learning to take place, that is, for decreasing the error below the average deviation of the payoff signal to perceive an initial fitness signal towards higher accuracy. In consequence, the subspace size requires the generation of classifiers with condition volumes of maximally that size, consequently yielding a *schema bound* on the population size to be able to cover the full problem space with such condition volumes. Finally, better classifiers with a certain condition volume need to be able to grow, that is, have reproductive opportunities before deletion can be expected, consequently yielding a *reproductive opportunity bound*.

Together these bounds give estimates on the necessary initial condition volumes and the resulting maximal population size necessary to cover a problem space. For example, given the need for an initial classifier volume of 0.01 of the encountered problem space, the population size N should be set to about $10/0.01 = 1000$ to assure proper rule evolution. Given that these factors are satisfied, better classifiers are assured to be identified and to grow in the population with high probability. For binary and for real-valued problem domains, these considerations have been quantified, showing that XCS is an approximate polynomial-time learning algorithm in problem domains with bounded complexity [47.25, 35].

The considerations above ensure the theoretic growth of better classifiers. However, the evolutionary component may still destroy relevant classifier structures due to mutation and crossover. Thus, neither mutation nor crossover may be overly disruptive. In extreme cases, where highly unstructured subspaces may need to be identified and recombined, estimation of distribution algorithms can help to identify these subspaces [47.31, 36]. In most cases, though, a sufficiently low mutation rate and uniform crossover suffice to learn successfully. However, clearly mutation is mandatory to detect more accurate classifier structures over time. Thus, a good compromise is necessary to ensure that offspring is usually mutated but its structure is not fully destructed. In the binary domain, for example, the mutation probability is consequently often set to $1/l$, where l is the number of bits of a problem instance. This is a typical choice for the mutation strength used in genetic algorithms – essentially setting the expected number of attributes that will be mutated to one.

47.2.3 When and How to Apply XCS

From the reflections above it becomes clear that XCS is designed to learn the target function of a problem by a population of locally accurate predictors, that is, classifiers. This target function may be the Q-value function in RL problems, a *correctness* function in classification problems, or also any other type of function. XCS is best suited to be applied in problem domains that can be partitioned into subspaces within which simple predictions yield accurate values. Moreover, XCS is even better suited to be applied to problems where regularities in the target function can be well-represented in classifier conditions, that is, subspaces in which the

function values are approximately equal should be compactly representable with few classifiers. Overall, XCS thus strives to develop distributed problem solutions in the form of a set of locally partially overlapping classifier structures, which cover the whole sampled problem space in a generalized way.

As long as a condition representation can be chosen that identifies expectable regularities in a data set or also in a reinforcement learning problem well, XCS is a good candidate to optimize these local condition structures iteratively online. However, also in offline, data mining-based classification problems XCS was applied successfully and it was shown that the generalization and accuracy performance XCS yield is comparable to other state-of-the art machine learning algorithms [47.25, 37], such as decision tree learners, instance-based classifiers, or support vector machines. Thus, XCS may be applied to multi-step Q-learning problems but also to single-step classification problems and general regression problems. Online generalization and optimal condition structuring for accurate predictions are the major features of XCS. From a regression perspective, XCS is a non-parametric regression algorithm that strives to minimize the expected absolute function approximation error, or also the expected squared function approximation error as put forward elsewhere [47.24].

The two components, (a) gradient-based rule prediction approximation and evaluation and (b) evolutionary rule structure evolution, are the key to successful XCS applications. With respect to rule structure evolution, also the XCS system strongly depends on distance representations, which can be compared with general kernel representations as used in support vector machines and elsewhere [47.38, 39]. As long as the represented kernel-based condition structures can be meaningfully modified by genetic operators, evolution and thus also XCS can be applied. Meanwhile, also sensible value predictions need to be generated. Gradient-based methods work best to approximate these predictions, whether the prediction is a single value, is computed linearly or polynomially from input, or its structured otherwise depends on the problem at hand and the gradient-based approximation approach available. The more the prediction structure fits with the regularities in the target function, the faster and more robust learning can be expected. While such structural considerations can improve system performance, the successful applications of XCS to various problem domains show that successful learning is usually not precluded by suboptimal structural choices.

47.2.4 Parameter Tuning in XCS

While XCS does, indeed, specify many parameters, only few parameters are really crucial. All other parameter values can typically be set to standard values. Here we discuss some rules of thumb for tuning the critical parameter settings and also provide standard settings. While the following recommendations have not been published elsewhere so far, they can be derived from observations and other recommendations found in the literature [47.25, 35, 40].

The two most important parameters are the maximal population size N and the strived-for error threshold ε_0. The larger the population size N is, the more capacity XCS has for learning and thus the more complex problems XCS can learn. On the other hand, the larger N is, the slower XCS learns, because it reproduces and deletes only two classifiers in a typical learning iteration. Parameter ε_0 specifies the targeted approximation error. In continuous function approximation problems, smaller ε_0 values demand finer problem space partitionings and thus larger population sizes to cover the whole problem space and to enable reproductive opportunities (see above). Moreover, ε_0 can partially determine the fitness signal available to XCS: if ε_0 is chosen very small, (47.4) will yield values very close to zero for all highly inaccurate classifiers. Thus, overly small ε_0 values should be avoided. In noisy problems, ε_0 should thus also not be chosen much smaller than the standard deviation of the noise expected in the function value signal.

Without much knowledge of a problem, one may start with a rather small population size N – say 1000 – and evaluate learning progress in this setting with a desired ε_0. If the generated approximation error over time does not decrease, then ε_0 should be set to about $1/10$ of the encountered error. Next, the population size N should be progressively increased, for example, to $N = 5000$ or more. If still no error decrease is observed, further analysis is necessary. If the population is filled with classifiers but the match set sizes are very small (below 5), better classifiers probably do not receive enough reproductive opportunities. In this case, first the initial condition volume should be increased – for example, in the binary domain the probability $P_\#$ would need to be increased (up to close to 1). If the match set still decreases to sizes below 5, the problem is rather hard, requiring a further population size increase. On the other hand, if the match set sizes are very large (above 100), then over-generalization takes place and XCS apparently does not pick up the fitness signal. In this case,

the initial condition volume should be decreased. If this does not help, then the GA application rate should be decreased to enforce a more accurate classifier evaluation before evolution applies. This can be accomplished by increasing the threshold θ_{GA} to say 100, 500, or even higher. An increase in θ_{GA} can also be crucial in problems where the problem domain is sampled highly unevenly, as is studied in detail elsewhere [47.29].

Several other parameter settings may be checked as well; the mutation rate should not be set overly high. As stated above, in the binary problem domain, for example, a mutation rate of $\mu = 1/l$, where l denotes the condition size, is a good rule of thumb. Crossover can mostly be applied without restrictions ($\chi = 1.0$) – especially when tournament selection for reproduction is chosen because in this case disruption is often prevented by choosing two equal classifiers. Other parameters can be safely set to somewhat standardized values. A typical initial parameter setting for XCS is: $N = 1000$, $\varepsilon_0 = 0.1$, $\mu = 1/l$, $\chi = 1$, $\alpha = 1$, $\beta = 0.2$, $\nu = 5$, $\theta_{GA} = 25$, $\gamma = 0.9$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, $P_{\#} = 0.5$, and $\tau = 0.4$.

47.3 XCSF

The XCS classifier system for real-valued inputs was introduced by *Wilson* in 1999, introducing Michigan-style LCSs to the real-valued problem domain [47.41, 42]. It was further enhanced to approximate continuous real-valued function surfaces in 2001/2002 [47.43], yielding an iterative online learning non-parametric regression system. XCS for function approximation (XCSF) essentially enhances and modifies XCS by changing its classifier condition structure to accept real-valued input. Moreover, the prediction part no longer predicts single values, but it computes its prediction from the input using linear approximation techniques, such as recursive least squares (RLS) [47.44]. Finally, the action part of the system is removed, applying the parts of the algorithm that were previously applied to $[A]$ to the match set $[M]$ in XCSF. Figure 47.4 illustrates the iterative learning process in XCSF.

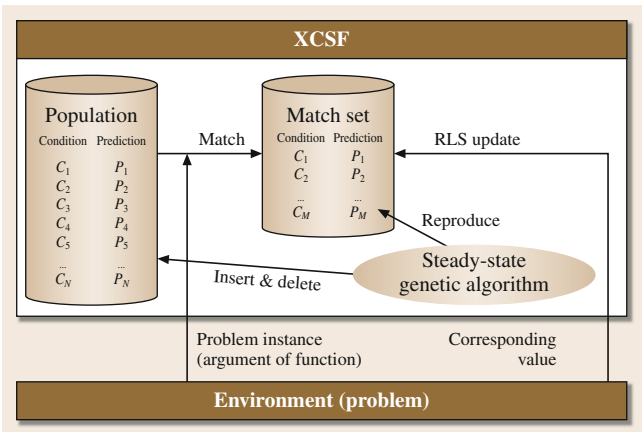
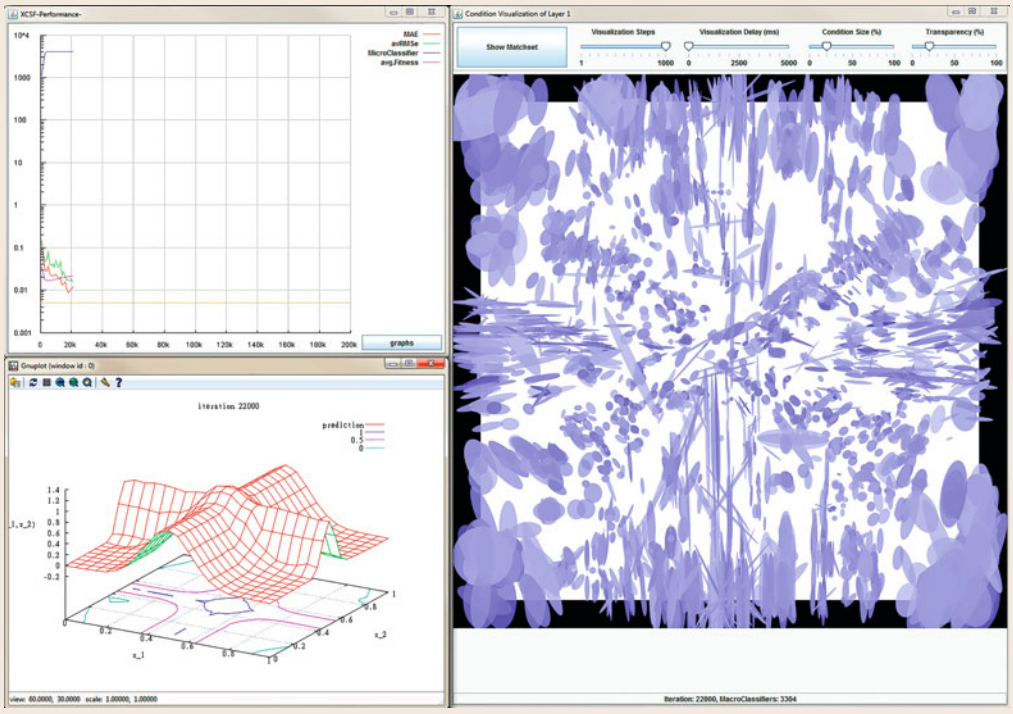


Fig. 47.4 The XCSF classifier system learns linear value predictions and usually does not specify actions. The feedback is the actual function value, which is used to update the linear approximators of the matching classifiers. The consequent error and fitness estimation updates are then considered in the evolutionary component for further optimization of the condition structures

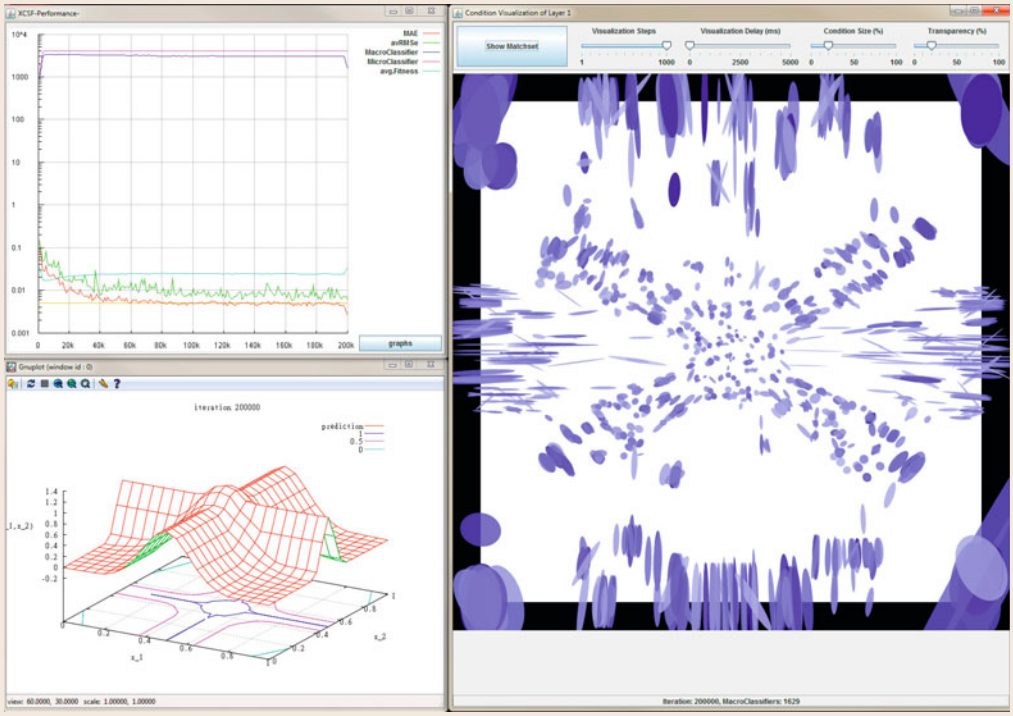
XCSF is thus a regression system that solves function approximation problems by developing partially overlapping locally weighted projections in the form of a population of classifiers. In this form, XCS develops problem solutions that are similar to those developed by the locally-weighted projection regression algorithm (LWPR), which is rather well-known in the robotics community [47.45]. A comparative study has shown that XCSF can outperform LWPR in various problem domains [47.46], often yielding better problem space partitionings, as well as more accurate function value approximations with a comparable number of individual locally linear approximators (i. e., classifiers). In XCSF, each classifier specifies in its condition the subspace within which it is applicable. Thus, the condition may be compared with a receptive field determining the neural activity of the classifier. Moreover, each classifier specifies a linear approximator weighted within its subspace. In effect, the function approximation problem is approximated by locally-weighted, overlapping linear approximations. While typically the weighting is

Fig. 47.5a,b Screenshots of the XCSF program learning to approximate the crossed ridge function. Current performance values are plotted on the *top left*. The current approximation surface is approximated on the *bottom left*. On the *right-hand side* the classifier condition structures are plotted. For visualization purposes, the receptive field sizes are plotted smaller than their actual size. *Darker* classifier conditions have higher fitness values ►

a)


XCSF-Performance-
MAE
avRMSe
MacroClassifier
avg.Fitness
graphs
Gnuplot (window id : 0)
Iteration 22000
prediction
Condition Visualization of Layer 1
Show Matchset
Visualization Steps
Visualization Delay (ms)
Condition Size (%)
Transparency (%)
Iteration: 22000, MacroClassifiers: 3304

b)


XCSF-Performance-
MAE
avRMSe
MacroClassifier
MicroClassifier
avg.Fitness
graphs
Gnuplot (window id : 0)
Iteration 200000
prediction
Condition Visualization of Layer 1
Show Matchset
Visualization Steps
Visualization Delay (ms)
Condition Size (%)
Transparency (%)
Iteration: 200000, MacroClassifiers: 1629

fitness-dependent, also a weighting based on the distance to the center of the classifier condition can be applied.

With this structure it has been shown that XCSF is very well suited for developing any type of kernel structure [47.47]. In effect, various condition structures have been applied, including rectangular structures with and without rotation and with various forms of representation [47.35, 48, 49]. Moreover, the linear approximations may be enhanced to polynomial approximations and others [47.50]. Finally, it is also possible to cluster a contextual space with conditions, while approximating linear (or other) predictions given totally different inputs. For example, the velocity kinematics of an arm can be predicted locally dependent on the angular arm constellation for redundancy resolution [47.51] (see further details below). Thus, XCSF is a highly flexible system with which other modifications in the condition and prediction parts of the classifiers may still yield highly vital system applications.

As an example, we applied XCSF to the *crossed ridge* function – a function that has been used as a benchmark in the neural computation and machine learning community for many years [47.45, 52]. The function contains a mix of linear and non-linear subspaces. It is specified in two dimensions as follows

$$f_1(x_1, x_2) = \max\left\{\exp(-10x_1^2)\,, \exp(-50x_2^2)\,, \right. \\ \left. 1.25\exp(-5(x_1^2 + x_2^2))\right\}\,. \qquad (47.8)$$

We ran XCSF with a maximum population size $N = 4000$ and a target error $\epsilon_0 = 0.005$ on this function, applying a condensation mechanism late in the run. Figure 47.5 shows that XCSF is able to yield a good function approximation very early in the run. The evolving classifier structures learn to suitably partition the problem space into local subspaces. In consequence, a smooth overall approximation surface is generated. Note how the inverse exponential hill in the center is approximated with nearly circular receptive fields, while the fields are selectively elongated in the x_1 or x_2 dimension due to the non-linearities caused by the ridges extending to the four sides. Towards the corners of the input space, the function flattens out so that the receptive fields become increasingly wider.

47.4 Data Mining

Data mining is a rather large field of research that generally addresses the challenge of extracting knowledge from data. In the LCS realm, the addressed data usually consists of a set of data instances, where each instance specifies a set of features and a corresponding class the data instance belongs to. LCSs then typically learn to mine the data by predicting the class likelihoods of unseen data instances, as well as by identifying the most relevant features and feature interactions for classification. Particularly Pitt-style LCSs have proven to be highly valuable in data mining applications. However, also the XCS classifier system was successfully applied in this domain.

The XCS system was also converted to an offline learning system; the sUpervised classifier system (UCS) algorithm [47.53] determines classifier predictions and resulting fitness values in a supervised manner. Meanwhile, the other learning aspects of UCS were derived from XCS. Both, XCS and UCS have shown effective if not even superior prediction accuracies in various data mining tasks – most of them taken from the UCI machine learning database repository [47.54]. When applying always the same standard setting and comparing with various other decision making algorithms, such as support-vector machines, decision tree learning, naive Bayes classifiers, and others implemented in the WEKA machine learning tool [47.55], XCSF outperformed these competing techniques in many cases – often depending on the problem at hand [47.25]. A similar performance was achieved with UCS, outperforming XCS in some cases due to its more accurate classifier prediction estimates. XCS was also further enhanced to be able to deal with highly unbalanced datasets in data mining domains by automatically adjusting the threshold that controls the frequency of GA applications θ_{GA} [47.29].

Pitt-style LCSs have been evaluated and applied to data mining problems even more extensively. The typical offline-learning scenario faced in data mining particularly suits the Pitt approach. However, also the fact that often very compact rule sets are strived for is advantageous for the Pitt approach. More than 10 years ago, the GALE architecture [47.56, 57] yielded very good performance results on a collection of datasets from the UCI repository. GALE distributes its evolutionary process adding additional niching biases due to a grid-based spatial distribution of individuals. A comparative study of GALE, XCS,

and other machine learning algorithms can be found in [47.58].

The GAassist architecture [47.59, 60] develops a priority list of classification rules. The advantage of GAassist is its developing compactness. A comparative analysis with XCS is provided in [47.61]. Later, the architecture was enhanced with ensemble learning techniques [47.62] and memetic algorithms [47.63], proving high scalability and fast learning of very compact rule sets.

Recently, many efficiency enhancement techniques from the GA literature (cf. [47.64]) and from other fields, including bioinformatics and systems biology [47.65] were applied to various LCSs. These techniques can help tremendously to improve the learning speed of LCSs, particularly in data mining realms. For example, *windowing techniques* select subsets of data instances to speed up the classifier evaluation process. *Fitness surrogates* were used to make the fitness estimation even cheaper [47.66]. *Hybrid methods* were already mentioned above; they combine traditional GA operators with informed ones, as is done when applying memetic algorithms, which locally improve the developing classifier structures when applied to LCSs. In combination, such techniques can yield LCSs that not only produce highly accurate classification performance and good generalizations, but they also offer solution interpretability allowing mining of the knowledge developed in the LCS rules, and they generate these results without requiring much computational time – which is often comparable to the time needed by much simpler machine learning techniques.

47.5 Behavioral Learning

While the application of LCSs to data mining problems will certainly still produce many further impressive results and promises to yield novel, deep insights into data structures, LCSs were originally designed as cognitive systems. Thus, in the following we will focus on LCSs as cognitive systems, their structures, and their potential as neural cognitive models. As had been sketched out above, the XCS classifier system in particular was compared with Q-learning in RL. We start from this perspective and detail various successful applications of XCS in reinforcement learning problems. Next, ALCSs are surveyed. ALCSs learn generalized cognitive maps that are suitable to apply Sutton's Dyna algorithm and value iteration techniques in general. A strong relation to factored RL techniques was pointed out recently in this respect [47.67]. Finally, robotics applications of LCSs are discussed and their potential is revealed.

47.5.1 Reward-Based Learning with LCSs

From the beginning [47.2] a big appeal to LCSs lay in the fact that they are designed for reward-based learning. Once the original bucket-brigade algorithm was replaced by Q-learning techniques, a theory developed in the RL community also applied to LCSs to a certain extent.

In XCS, in particular, it was shown that the system approximates the Q-value function by a collection of classifiers. The prediction array (47.1) calculation essentially approximates the current Q-value estimates for the current state in the environment. The fitness weighting based on the relative accuracies, which are normalized to one, assures that these Q-value estimates on average do not over or underestimate the expected Q-value. Moreover, since Q-learning is an off-policy learning technique, XCS is well-suited to be combined with it because also XCS benefits from exploring all possible state–action combinations in the long run – striving to develop an approximation of the complete Q-value function in the problem space.

As a result, XCS has been successfully applied to learning optimal paths in various maze environments. Starting from the Woods1 and Woods2 environments proposed by *Wilson* [47.20, 21], XCS's performance and generalization capabilities have been investigated in various mazes [47.68]. For illustrative purposes such mazes are shown in Fig. 47.6. These maze environments provide information about the surrounding grid cells, indicating whether they are either free or occupied by an obstacle or by food. Reaching the latter cell usually results in a reward trigger. Movements are typically possible to the eight surrounding cells, yielding a rather large action space. The point of providing sensory state information rather that cell IDs or coordinates is that XCS is then able to exhibit its generalization capabilities. It essentially manages to generalize over the sensory state space ignoring irrelevant bits and generalizing over the states with respect to state–action combinations that yield the same reward.

Performance in many of these environments has yielded extreme generalization capabilities. For exam-

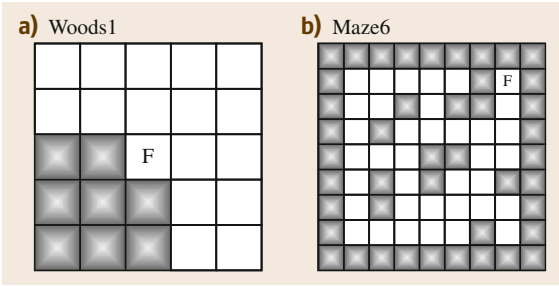


Fig. 47.6a,b Two highly typical maze environments used as benchmarks in the LCS literature for generalized reinforcement learning. Woods1 is a toroidal maze. In Maze6 the food location is much harder to find. In both cases, the LCS-controlled agent perceives information about the eight neighboring cells encoding free, blocked, and food cells by means of two bits. The agent can execute movements to each of these cells. Movements to blocked cells yield no reward. A movement to the food cell triggers reward and a reset of the agent

ple, in the Maze6 environment (Fig. 47.6) up to 90 irrelevant bits were introduced, which changed randomly while interacting with the environment. While learning was slightly delayed and a larger population size was needed for successful learning, the optimal Q-value function was still extracted from iterative interactions [47.25]. Thus, XCS learned the optimal Q-value function in a problem space that contained more than 10^{30} potential sensory state encodings. Also rather noisy action outcomes did not preclude learning success. Later, it was shown that highly effective generalizations are even possible when each bit in the sensory encoding is relevant. In [47.36] the encoding for each bit was changed to a nested Boolean function, such as the parity function. XCS was still able to learn the optimal Q-value function, while Q-learning without generalization failed miserably due to the large state space. Thus, XCS is able to identify those aspects of the available sensory information that are relevant for accurate reward predictions.

To successfully apply XCS in these scenarios, one crucial modification was necessary to stabilize the Q-values and thus the derived fitness values: the update of the classifier predictions had to be further modified by the error gradient factor, converting (47.3) to

$$r \leftarrow r + \beta(\rho - r)\frac{f}{\sum_{\mathrm{cl}\in[A_{-1}]}\mathrm{cl}.f} \,. \qquad (47.9)$$

The exact derivation of this equation can be found in the literature [47.69]. The gradient term essentially results in much more stable performance and successful learning and generalization in problems that require the establishment of long reward chains. It stabilizes the reward learning by down-scaling updates of inaccurate and unreliable classifiers. Consequently, these rules do not tend to over-estimate reward, and thus learning progress is stabilized. As a further consequence, XCS with gradient-based reward predictions updates was also successfully applied to blocks world problems, in which even more generalizations are possible [47.25].

The generalization capabilities of LCSs reached even as far as being successfully applied to control simple light following behavior on a real robot platform [47.70, 71]. In this case, however, reward learning was maximized and no complete Q-value function approximation developed. Nonetheless, this work constituted one of the first successful application in the robotics domain.

Besides condition–action Michigan-style LCSs, such as the XCS, other Michigan-style LCS techniques have been applied for behavioral learning and also for learning cognitive maps. Such anticipatory learning classifier systems are surveyed in the following.

47.5.2 Anticipatory Learning Classifier Systems

Anticipatory learning classifier systems (ALCSs) are learning systems that learn a generalized predictive model or *cognitive map* [47.72] of the encountered environment online. ALCSs are typical Michigan-style LCSs. However, in contrast to the usual classifier structure, classifiers in ALCSs have a state prediction or *anticipatory* part that predicts the environmental changes in the environment caused when executing the specified action in the specified context. As in XCS, ALCSs derive classifier fitness estimates from the accuracy of their predictions. However, the accuracy of the anticipatory state predictions are considered, rather than the accuracy of the reward prediction. Figure 47.7 illustrates the typical structures and learning processes that apply in an ALCS architecture.

Rick Riolo originally proposed an ALCS that generated its cognitive map mediated by a *message list* storage system, which was also used in Holland's original classifier system architecture [47.73]. However, this approach appeared to not be sufficiently elegant to enable any serious learning. Starting with *Stolzmann*'s anticipatory classifier system [47.74], various ALCS architectures were developed. Particularly in maze problems, optimal behavior was achieved with

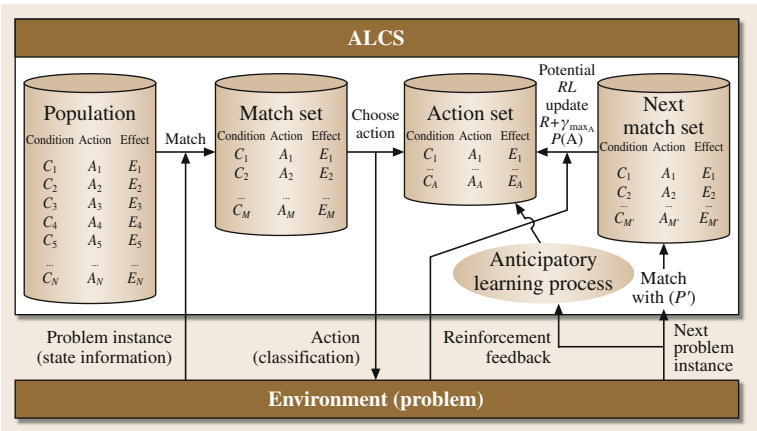


Fig. 47.7 Instead of the condition–action–reward prediction rules in typical LCSs, ALCSs encode and develop condition–action–effect rules. Typically, the structural optimization of these rules is done by a combination of evolutionary and heuristic techniques

various ALCSs [47.75–79]. To prevent the development of overgeneral models for concurrent reward learning, the reward learning process was often decoupled, yielding a system that learns a cognitive map based on LCS principles, and, additionally, a state value estimation system. In combination, DYNA-based learning techniques [47.80] were applied to improve the state value estimations also offline. These techniques allowed the simulation of animal-like behavioral patterns, such as reward adaptations based on knowledge about the behavioral consequences in rats in a T-maze environment [47.81], as well as in controlled devaluation or satiation experiments [47.82]. In these studies it was also pointed out that ALCSs do not only allow DYNA-based reward learning updates, but also enable the application of search and planning techniques for improving behavioral performance of the system. Even curiosity mechanisms have been added [47.83] to speed up the learning progress. Most approaches, however, never generalized the list of states with associated rewards.

The combination of the ACS2 system with the XCS system for state-value estimations, terming the resulting system XACS (x-anticipatory classifier system), may be the one with the most current potential for future research [47.84]. XACS essentially applies two LCS learning mechanisms: one being an ALCS architecture in the form of ACS2, which learns a cognitive model of the encountered environment, and the other one being the XCS system, which learns state-value estimations in this case. Figure 47.8 illustrates the components in the XACS architecture and their interactions.

XACS has been shown to develop optimal behavior in blocks world problems in which other approaches failed to yield proper generalizations and resultingly optimal behavior control. Moreover, the reward-based generalization mechanism in XACS is directly based on the XCS classifier system, thus enabling the incorporation of any tools and representations developed for XCS so far. The generalizations that were developed confirmed the identification of task-relevant perceptual attributes. In the XCS components, reward-distinguishing attributes were identified. In the ACS2 component, on the other hand, state prediction-relevant components were detected. In consequence, generalized detectors for prediction with respect to reward and state could be distinguished. The implementation of other anticipatory mechanisms in XACS, such as task-dependent attentional mechanisms, further interactions of the learning components, and multiple behavioral modules for the representation of multiple motivations (or needs) [47.84] are still open issues in the LCS realm. Further research with ALCSs is expected to yield highly promising, cognitive learning architectures.

47.5.3 Controlling a Robot Arm with an LCS

We end this section of behavioral learning with the XCSF system. Over the last decade or so it has become increasingly clear that XCS is extremely well suited to partition a contextual space for the generation of accurate predictions. Predictions, however, do not necessarily need to be reward predictions. Behavioral consequences serve just as well as a target for predictions. The forward kinematics mapping in the robotics domain [47.85] offers even another potential target for learning.

Consequently, XCSF was modified to learn the forward velocity kinematics of a robotic arm in simu-

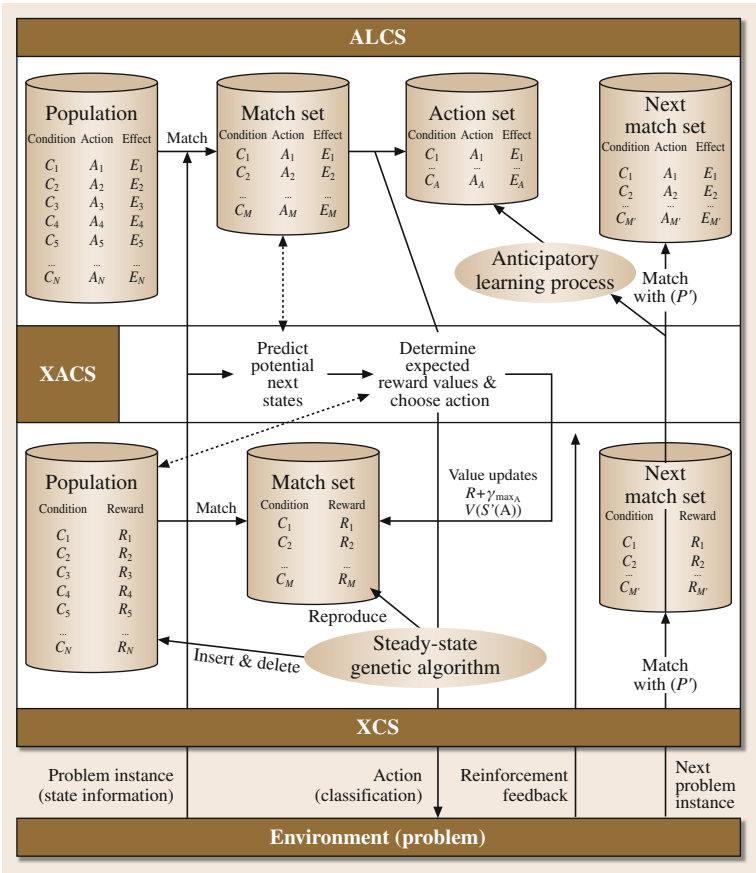


Fig. 47.8 The XACS system combines the model learning capabilities of ALCSs with the generalizing reinforcement learning capabilities of XCS. Consequently, generalizations in the two system components are targeted towards a compact representation for accurate predictions and for reward predictions, respectively. The combined system enables the application of lookahead planning and search techniques for behavioral control as well as of reinforcement learning techniques and combinations thereof

lation [47.86]. To do so, XCSF projects its condition parts into the joint angle space of the robotic arm. However, its locally linear predictions receive as input small joint movements, that is, changes in joint space and predict the consequent change in task space, that is, changes of the end-effector location. This mapping has the great advantage that it is locally linear so that given a current joint angle constellation of the arm not only location changes of joint angle movements can be predicted but also directional motion of the end-effector can be invoked by inverting the locally linear forward velocity mappings. Seeing that those are linear, the inversion can be rather easily done using linear algebra techniques. Given a redundant arm system – one that has more degrees of freedom (i. e., joint angles to manipulate) than actual locations to move to – it is possible to add additional constraints to the arm motion. For example, the arm can be driven to maintain a *relaxed* arm posture while pursuing a certain goal or it may be forced to prevent moving a certain joint angle at all [47.87]. Recent advancements in the exploration strategy, which can be self-induced by the XCSF controller during learning, have shown that XCSF is able to learn to control all seven degrees of freedom of a humanoid arm highly effectively – flexibly adhering to different constraints while pursuing motions to certain goal locations. Moreover, mappings could be learned in different reference frame representations. For example, end-effector locations were either represented in a Cartesian coordinate system or in a distance plus angles encoding. XCSF learned different classifier structures due to the differences in the linearities encountered. Nonetheless, XCSF yielded equally good arm control in both cases [47.51]. Figure 47.9 illustrates the XCSF setup for arm control.

These results confirmed that XCSF may very well be further developed into a cognitive system architecture for behavioral control. While this type of architecture was probably not the one envisioned by Holland originally, it may still prove highly valuable. Various

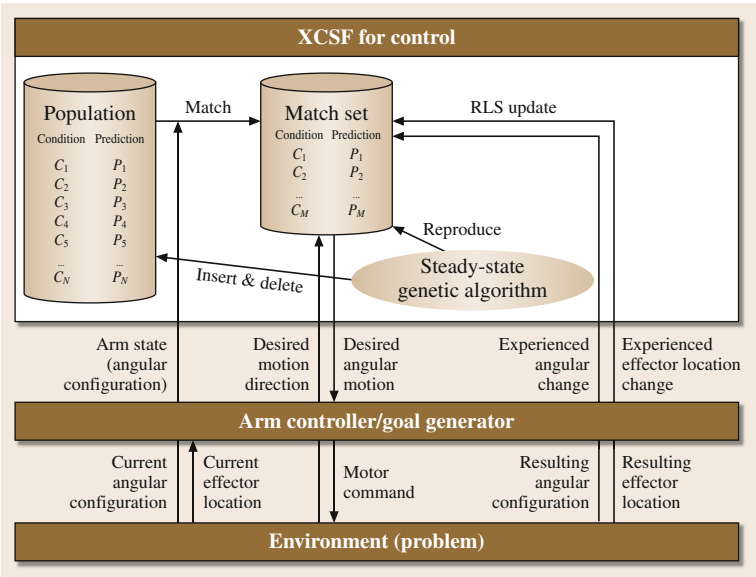


Fig. 47.9 In the published robot arm control applications, XCSF clusters the contextual configuration state of the arm and learns linear approximations of the average Jacobian in the respective subspaces. In consequence, the system can generate both forward predictions of movement consequences as well as inverse control commands when directional movements of the arm are desired

neuroscientific evidence points out that similar forward-inverse predictive-control structures may be found in the cerebellum [47.88, 89]. Only more detailed knowledge on cortical and cerebellar structures may allow the direct comparison of the shapes and orientations of the receptive fields developed by the XCSF system and potential cortical and neural structures found in the brain. While the brain may not implement actual evolutionary techniques literally, as XCSF does, it appears plausible that local competitions take place [47.90]. Moreover, it is known that neurons populate novel information sources once available – as XCSF does. Further research in neural computation with LCSs may prove highly valuable.

47.6 Conclusions

While LCSs have been applied to a wide variety of problems, still there are many potential developments that have not been further evaluated. In the following, potential future research directions are summarized.

At the moment nearly all LCSs are flat in that they develop one population of classifiers (or competing sets of classifiers in the Pitt-style system). All of the classifiers, however, apply to the same problem granularity. Ever since the introduction of LCSs by Holland, the development of *default hierarchies* was envisioned. However, so far it was never convincingly or rigorously accomplished [47.19]. Default hierarchies refer to classifier systems in which general rules predict one thing but more specialized rules predict exceptions of the general rule. The emergent development of default hierarchies in LCSs remains an open challenge.

With the most recent understanding of LCSs and the XCS system in particular, it seems that at least the development of a hierarchically-structured LCS architecture is within our grasp. We expect such a hierarchical LCS to progressively refine its predictions in a hierarchical way. Default rules may gain a certain level of accuracy, but more specialized rules may identify exceptions of the default prediction. Alternatively, the more specialized rules may also simply add further accuracy to the default predictions where and when necessary. In the latter case, a hierarchical predictive system may develop that allows the progressive refinement of activated predictions until the finest prediction granularity in the hierarchical representation is reached.

When developing hierarchical LCSs, also network LCSs seem to be of vital importance. For example, when developing classifier structures in spatial domains that are intricately structured, a network structure may provide additional hints on the connectivity of

the space. Especially the case where XCSF learns velocity kinematics, or generally, contextually-dependent sensory-motor contingencies – as sketched out in the section on controlling a robot arm with XCSF above – a network structure can give additional hints on how the sensorimotor space is structured and may be traversed. Networks of LCS classifiers may allow the application of lookahead planning and goal-oriented control – as was pursued in early work in [47.91].

A network structure may also enable the speed-up of the XCS matching process. For example, when a problem space is sampled by means of a random walk process, overlapping classifiers may be directly identified within a classifier structure instead of applying a global matching process in each iteration. Also, when XCSF is used for goal-directed control – as mentioned above with respect to velocity kinematics – this may improve the efficiency of the system tremendously. Furthermore, given a hierarchically network structured LCS system matching may proceed from coarse-to-fine-grained levels. All these processes may speed up the matching, which is often considered a bottleneck in LCS research and has been improved by means of numerous approaches over the recent years [47.92, 93].

Besides these additions, also ALCSs may be pursued further, as sketched out above. From a cognitive modeling perspective, ALCSs essentially learn generalized schemata or production rules [47.94–96], which specify the expected state changes perceived after the execution of the specified action. Such rules may be applied by the cognitive science community for learning, for example, ACT-R structures [47.97]. The lookahead planning capabilities, the sensorimotor generalization capabilities, as well as the abstraction capabilities of these systems still ask for further development. The recent point that ALCSs can be very effectively applied to factored RL problems [47.98] should be further pursued. Also, the combination of ALCS-based cognitive map or concept learning and XCS-based reward learning promises further research advancements.

Even without the addition of hierarchies, network structures, or anticipations, however, LCSs can be successfully applied to various domains including reinforcement learning problems, classification and data mining problems, and regression problems. XCS, in particular, learns iteratively online, striving for the development of a compact, maximally general, and maximally accurate problem solution. Pitt-style systems typically learn offline and are thus most promising in large-scale data mining tasks in which rather small compact sets of rules are searched for. Seeing that the learning mechanisms of LCSs are highly flexible, it is possible to substitute the condition of a classifier with any other form or condition structure, as long as this structure can be mutated and recombined in a way that small structural changes also yield small changes in the defined subspace within which the condition matches. Similarly, the prediction structure can be replaced with any other prediction structure that can be quickly and accurately adapted by suitable learning techniques. Thus, the available LCS techniques – such as GALE and GAassist on the Pitt side and XCS, XCSF, or XACS on the Michigan side – can be further exploited and combined with novel structures and forms of representations. Learning promises to be robust due the combination of a flexible evolutionary component, which searches for optimal rule structures, and the gradient-based fitness estimation, which quickly yields useful prediction and fitness estimations. It seems only a matter of time until LCSs gain even more recognition and be successfully applied to even more diverse problem domains and challenging research tasks.

47.7 Books and Source Code

Further information about learning classifier systems can be found in the biannually published IWLCS (International Workshop on Learning Classifier Systems) workshop proceedings and yearly workshops on the topic. A book on LCSs and the XCS classifier system in particular covers XCS from a theoretical and application-oriented point of view and also provides a detailed algorithmic description of the system [47.25]. A more theoretical coverage of the approximation approach in XCS can be found in [47.23]. Several books also give further details on theoretical considerations [47.23, 99] as well as on successful applications of LCSs [47.100, 101]. The source code can be found online, for example, for XCS in C++ [47.102] as well as for XCSF in Java [47.103].

References

47.1 J.H. Holland: *Adaptation in Natural and Artificial Systems* (Univ. of Michigan, Ann Arbor 1975)

47.2 J.H. Holland: Adaptation. In: *Progress in Theoretical Biology*, Vol. 4, ed. by R. Rosen, F.M. Snell (Academic, New York 1976) pp. 263–293

47.3 L.B. Booker, D.E. Goldberg, J.H. Holland: Classifier systems and genetic algorithms, Artif. Intell. **40**, 235–282 (1989)

47.4 J.H. Holland, J.S. Reitman: Cognitive systems based on adaptive algorithms. In: *Pattern Directed Inference Systems*, ed. by D.A. Waterman, F. Hayes-Roth (Academic, New York 1978) pp. 313–329

47.5 L.P. Kaelbling, M.L. Littman, A.W. Moore: Reinforcement learning: A survey, J. Artif. Intell. Res. **4**, 237–285 (1996)

47.6 R.S. Sutton, A.G. Barto: *Reinforcement Learning: An Introduction* (MIT Press, Cambridge 1998)

47.7 J.H. Holland: Properties of the bucket brigade algorithm, Proc. Int. Conf. Genet. Algorithms Appl. (1985) pp. 1–7

47.8 D.E. Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison-Wesley, Reading 1989)

47.9 S.F. Smith: A learning system based on genetic adaptive algorithms, Ph.D. Thesis (Univ. of Pittsburgh, Pittsburgh 1980)

47.10 K.A. De Jong: An analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis (Univ. of Michigan, Ann Arbor 1975)

47.11 L.B. Booker: Intelligent behavior as an adaptation to the task environment, Ph.D. Thesis (The Univ. of Michigan, Ann Arbor 1982)

47.12 S.W. Wilson: Knowledge growth in an artificial animal, Proc. Int. Conf. Genet. Algorit. Appl. (1985) pp. 16–23

47.13 S.W. Wilson: Classifier systems and the animat problem, Mach. Learn. **2**, 199–228 (1987)

47.14 D.E. Goldberg: Computer-aided gas pipeline operation using genetic algorithms and rule learning, Diss. Abstr. Int. **44**, 3174B (1983)

47.15 K.A. De Jong: Learning with genetic algorithms: An overview, Mach. Learn. **3**, 121–138 (1988)

47.16 K.A. De Jong, W.M. Spears, D.F. Gordon: Using genetic algorithms for concept learning, Mach. Learn. **13**, 161–188 (1993)

47.17 R.L. Riolo: Bucket brigade performance: I. Long sequences of classifiers, Proc. 2nd Int. Conf. Genet. Algorithms (ICGA87), ed. by J.J. Grefenstette (Lawrence Erlbaum Associates, Cambridge 1987) pp. 184–195

47.18 R.E. Smith, H. Brown Cribbs: Is a learning classifier system a type of neural network?, Evol. Comput. **2**, 19–36 (1994)

47.19 J.H. Holland, L.B. Booker, M. Colombetti, M. Dorigo, D.E. Goldberg, S. Forrest, R.L. Riolo, R.E. Smith, P.L. Lanzi, W. Stolzmann, S.W. Wilson: What is a learning classifier system?, Lect. Notes Comput. Sci. **1813**, 3–6 (2000)

47.20 S.W. Wilson: ZCS: A zeroth level classifier system, Evol. Comput. **2**, 1–18 (1994)

47.21 S.W. Wilson: Classifier fitness based on accuracy, Evol. Comput. **3**, 149–175 (1995)

47.22 C.J.C.H. Watkins: Learning from delayed rewards, Ph.D. Thesis (King's College, Cambridge 1989)

47.23 J. Drugowitsch: Design and Analysis of Learning Classifier Systems: A Probabilistic Approach, Studies in Computational Intelligence (Springer, Berlin, Heidelberg 2008)

47.24 J. Drugowitsch, A. Barry: A formal framework and extensions for function approximation in learning classifier systems, Mach. Learn. **70**, 45–88 (2008)

47.25 M.V. Butz: *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design* (Springer, Berlin, Heidelberg 2006)

47.26 B. Widrow, M. Hoff: Adaptive switching circuits, West. Electron. Show Conv. **4**, 96–104 (1960)

47.27 P.-Y. Oudeyer, F. Kaplan, V.V. Hafner: Intrinsic motivation systems for autonomous mental development, IEEE Trans. Evol. Comput. **11**, 265–286 (2007)

47.28 G. Venturini: Adaptation in dynamic environments through a minimal probability of exploration, from animals to animats 3, Proc. 3rd Int. Conf. Simul. Adapt. Behav. (1994) pp. 371–381

47.29 A. Orriols-Puig, E. Bernadó-Mansilla, D.E. Goldberg, K. Sastry, P.L. Lanzi: Facetwise analysis of XCS for problems with class imbalances, IEEE Trans. Evol. Comput. **13**, 1093–1119 (2009)

47.30 M.V. Butz, K. Sastry, D.E. Goldberg: Strong, stable, and reliable fitness pressure in XCS due to tournament selection, Genet. Program. Evol. Mach. **6**, 53–77 (2005)

47.31 M.V. Butz, M. Pelikan, X. Llorà, D.E. Goldberg: Automated global structure extraction for effective local building block processing in XCS, Evol. Comput. **14**, 345–380 (2006)

47.32 S.W. Wilson: Generalization in the XCS classifier system, genetic programming 1998, Proc. 3rd Ann. Conf. (1998) pp. 665–674

47.33 M.V. Butz, T. Kovacs, P.L. Lanzi, S.W. Wilson: Toward a theory of generalization and learning in XCS, IEEE Trans. Evol. Comput. **8**, 28–46 (2004)

47.34 T. Kovacs: XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions. In: *Soft Computing in Engineering Design and Manufacturing*, ed. by R. Roy, P.K. Chawdhry, R.K. Pant (Springer, Berlin, Heidelberg 1997) pp. 59–68

47.35 P.O. Stalph, X. Llorà, D.E. Goldberg, M.V. Butz: Resource management and scalability of the XCSF learning classifier system, Theor. Comput. Sci. **425**, 126–141 (2012)

47.36 M.V. Butz, P.L. Lanzi: Sequential problems that test generalization in learning classifier systems, Evol. Comput. **2**, 141–147 (2009)

47.37 L. Bull, E. Bernadó-Mansilla, J. Holmes (Eds.): *Learning Classifier Systems in Data Mining*, Studies in Computational Intelligence, Vol. 125 (Springer, Berlin, Heidelberg 2008)

47.38 B. Schökopf, A.J. Smola: *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (MIT Press, Cambridge 2001)

47.39 W. Liu, J.C. Principe, S. Haykin: *Kernel Adaptive Filtering: A Comprehensive Introduction*, 1st edn. (Wiley, Hoboken 2010)

47.40 M.V. Butz, S.W. Wilson: An algorithmic description of XCS, Soft Comput. **6**, 144–153 (2002)

47.41 S.W. Wilson: Get real! XCS with continuous-valued inputs. In: *Festschrift in honor of John H. Holland*, ed. by L. Booker, S. Forrest, M. Mitchell, R.L. Riolo (Center for the Study of Complex Systems, Ann Arbor 1999) pp. 111–121

47.42 S.W. Wilson: Get real! XCS with continuous-valued inputs, Lect. Notes Comput. Sci. **1813**, 209–219 (2000)

47.43 S.W. Wilson: Classifiers that approximate functions, Nat. Comput. **1**, 211–234 (2002)

47.44 S. Haykin: *Adaptive Filter Theory*, 4th edn. (Prentice Hall, Upper Saddle River 2002)

47.45 S. Vijayakumar, A. D'Souza, S. Schaal: Incremental online learning in high dimensions, Neural Comput. **17**, 2602–2634 (2005)

47.46 P. Stalph, J. Rubinsztajn, O. Sigaud, M.V. Butz: Function approximation with LWPR and XCSF: A comparative study, Evol. Comput. **5**, 103–116 (2012)

47.47 M.V. Butz: Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system, Proc. Genet. Evol. Comput. Conf. (GECCO 2005) (2005) pp. 1835–1842

47.48 C. Stone, L. Bull: For real! XCS with continuous-valued inputs, Evol. Comput. **11**, 299–336 (2003)

47.49 M.V. Butz, P.L. Lanzi, S.W. Wilson: Function Approximation With XCS: Hyperellipsoidal Conditions, Recursive Least Squares, and Compaction, IEEE Trans. Evol. Comput. **12**, 355–376 (2008)

47.50 D. Loiacono, P.L. Lanzi: Recursive least squares and quadratic prediction in continuous multistep problems, Lect. Notes Comput. Sci. **6471**, 70–86 (2010)

47.51 P.O. Stalph, M.V. Butz: Learning local linear Jacobians for flexible and adaptive robot arm control, Genet. Program. Evol. Mach. **13**, 137–157 (2012)

47.52 S. Schaal, C.G. Atkeson: Constructive incremental learning from only local information, Neural Comput. **10**, 2047–2084 (1998)

47.53 E. Bernadó-Mansilla, J.M. Garrell-Guiu: Accuracy-based learning classifier systems: Models, analysis, and applications to classification tasks, Evol. Comput. **11**, 209–238 (2003)

47.54 K. Bache, M. Lichman: *UCI Machine Learning Repository* (Univ. of California, School of Information and Computer Sciences 2013) http://archive.ics.uci.edu/ml

47.55 I.H. Witten, E. Frank: *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations* (Morgan Kaufmann, San Francisco 2000)

47.56 X. Llorà, J.M. Garrell: Knowledge independent data mining with fine-grained parallel evolutionary algorithms, Proc. Genet. Evol. Comput. Conf. (GECCO 2001) (2001) pp. 461–468

47.57 X. Llorà, J.M. Garrell: Inducing partially-defined instances with evolutionary algorithms, Proc. 18th Int. Conf. Mach. Learn. (ICML 2001) (2001)

47.58 E. Bernadó, X. Llorà, J.M. Garrell: XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks, Lect. Notes Comput. Sci. **2321**, 115–132 (2002)

47.59 J. Bacardit, J.M. Garrell: Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system, Lect. Notes Comput. Sci. **2724**, 1818–1831 (2003)

47.60 J. Bacardit, M.V. Butz: Data mining in learning classifier systems: Comparing XCS with GAssist, Lect. Notes Comput. Sci. **4399**, 282–290 (2007)

47.61 J. Bacardit, M.V. Butz: *Data mining in learning classifier systems: Comparing XCS with GAssist* (IlliGAL, Univ. of Illinois at Urbana-Champign 2004)

47.62 J. Bacardit, N. Krasnogor: Empirical evaluation of ensemble techniques for a Pittsburgh learning classifier system, Lect. Notes Comput. Sci. **4998**, 255–268 (2008)

47.63 J. Bacardit, N. Krasnogor: Performance and efficiency of memetic Pittsburgh learning classifier systems, Evol. Comput. **17**, 307–342 (2009)

47.64 K. Sastry, D.E. Goldberg, X. Llorá: Towards billion-bit optimization via a parallel estimation of distribution algorithm, Proc. Genet. Evol. Comput. Conf. (GECCO 2007) (2007) pp. 577–584

47.65 J. Bacardit, E. Burke, N. Krasnogor: Improving the scalability of rule-based evolutionary learning, Memet. Comput. **1**, 55–67 (2009)

47.66 X. Llorà, K. Sastry, T.-L. Yu, D.E. Goldberg: Do not match, inherit: Fitness surrogates for genetics-based machine learning techniques, Proc. Genet. Evol. Comput. Conf. (GECCO 2007) (2007) pp. 1798–1805

47.67 O. Sigaud, M.V. Butz, O. Kozlova, C. Meyer: *Anticipatory Learning Classifier Systems and Factored Reinforcement Learning* (Springer, Berlin, Heidelberg 2009) pp. 321–333

47.68 P.L. Lanzi: An analysis of generalization in the XCS classifier system, Evol. Comput. **7**, 125–149 (1999)

47.69 M.V. Butz, D.E. Goldberg, P.L. Lanzi: Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems, IEEE Trans. Evol. Comput. **9**, 452–473 (2005)

47.70 J. Hurst, L. Bull: Self-adaptation in classifier system controllers, Artif. Life Robot. **5**, 109–119 (2001)

47.71 J. Hurst, L. Bull: A neural learning classifier system with self-adaptive constructivism for mobile robot learning, Artif. Life **12**, 1–28 (2006)

47.72 E.C. Tolman: Cognitive maps in rats and men, Psychol. Rev. **55**, 189–208 (1948)

47.73 R.L. Riolo: Lookahead planning and latent learning in a classifier system, from animals to animats, Proc. 1st Int. Conf. Simul. Adapt. Behav. (1991) pp. 316–326

47.74 W. Stolzmann: Anticipatory classifier systems, Genetic Programming 1998, Proc. 3rd Ann. Conf. (1998) pp. 658–664

47.75 M.V. Butz: *Anticipatory Learning Classifier Systems* (Kluwer, Boston 2002)

47.76 M.V. Butz, D.E. Goldberg, W. Stolzmann: The anticipatory classifier system and genetic generalization, Nat. Comput. **1**, 427–467 (2002)

47.77 P. Gérard, O. Sigaud: YACS: Combining dynamic programming with generalization in classifier systems, Lect. Notes Comput. Sci. **1996**, 52–69 (2001)

47.78 P. Gérard, J.-A. Meyer, O. Sigaud: Combining latent learning and dynamic programming in MACS, Eur. J. Oper. Res. **160**, 614–637 (2005)

47.79 W. Stolzmann, M.V. Butz: Latent learning and action planning in robots with anticipatory classifier systems, Lect. Notes Comput. Sci. **1813**, 301–317 (2000)

47.80 R.S. Sutton: DYNA: an integrated architecture for learning, planning, and reacting, ACM SIGART Bull. **2**(4), 160–163 (1991)

47.81 W. Stolzmann, M.V. Butz, J. Hoffmann, D.E. Goldberg: First cognitive capabilities in the anticipatory classifier system, from animals to animats 6, Proc. 6th Int. Conf. Simul. Adapt. Behav. (2000) pp. 287–296

47.82 M.V. Butz, J. Hoffmann: Anticipations control behavior: Animal behavior in an anticipatory learning classifier system, Adapt. Behav. **10**, 75–96 (2002)

47.83 M.V. Butz: Biasing exploration in an anticipatory learning classifier system, Lect. Notes Comput. Sci. **2321**, 3–22 (2002)

47.84 M.V. Butz, D.E. Goldberg: Generalized state values in an anticipatory learning classifier system. In: *Anticipatory Behavior in Adaptive Learning Systems: Foundations, Theories, and Systems*, ed. by M.V. Butz, O. Sigaud, P. Gérard (Springer, Berlin, Heidelberg 2003) pp. 282–301

47.85 B. Siciliano, O. Khatib: *Springer Handbook of Robotics* (Springer, Berlin, Heidelberg 2007)

47.86 M.V. Butz, O. Herbort: Context-dependent predictions and cognitive arm control with XCSF, Proc. Genet. Evol. Comput. Conf. (GECCO 2008) (2008) pp. 1357–1364

47.87 M.V. Butz, G.K.M. Pedersen, P.O. Stalph: Learning sensorimotor control structures with XCSF: Redundancy exploitation and dynamic control, Proc. Genet. Evol. Comput. Conf. (GECCO 2009) (2009) pp. 1171–1178

47.88 D.M. Wolpert, R.C. Miall, M. Kawato: Internal models in the cerebellum, Trends Cogn. Sci. **2**, 338–347 (1998)

47.89 J.G. Fleischer: Neural correlates of anticipation in cerebellum, basal ganglia, and hippocampus, Lect. Notes Comput. Sci. **4520**, 19–34 (2007)

47.90 C.T. Fernando, E. Szathmary, P. Husbands: Selectionist and evolutionary approaches to brain function: A critical appraisal, Front. Comput. Neurosci. **6**, doi: 10.3389/fncom.2012.00024 (2012)

47.91 A. Tomlinson, L. Bull: A corporate XCS, Lect. Notes Comput. Sci. **1813**, 195–208 (2000)

47.92 X. Llorà, K. Sastry: Fast rule matching for learning classifier systems via vector instructions, Proc. Genet. Evol. Comput. Conf. (GECCO 2006) (2006) pp. 1513–1520

47.93 M.V. Butz, P.L. Lanzi, X. Llorà, D. Loiacono: An analysis of matching in learning classifier systems, Proc. Genet. Evol. Comput. Conf. (GECCO 2008) (2008) pp. 1349–1356

47.94 J.R. Anderson: *Rules of the Mind* (Lawrence Erlbaum Associates, Hillsdale 1993)

47.95 G.L. Drescher: *Made-Up Minds: A Constructivist Approach to Artificial Intelligence* (MIT Press, Cambridge 1991)

47.96 A. Newell: Physical symbol systems, Cogn. Sci. **4**, 135–183 (1980)

47.97 J.R. Anderson, D. Bothell, M.D. Byrne, S. Douglass, C. Lebiere, Y. Qin: An integrated theory of the mind, Psychol. Rev. **111**, 1036–1060 (2004)

47.98 O. Sigaud, S. Wilson: Learning classifier systems: A survey, soft computing – a fusion of foundations, Methodol. Appl. **11**, 1065–1078 (2007)

47.99 L. Bull, T. Kovacs (Eds.): *Foundations of Learning Classifier Systems*, Stud. Fuzziness and Soft Comput, Vol. 183 (Springer, Berlin, Heidelberg 2005)

47.100 L. Bull (Ed.): *Applications of Learning Classifier Systems* (Springer, Berlin, Heidelberg 2004)

47.101 L. Bull: On lookahead and latent learning in Simple LCS, Learn. Classif. Syst. Int. Workshops, IWLCS 2006-2007, ed. by J. Bacardit, E. Bernad-Mansilla, M.V. Butz (Springer, Berlin, Heidelberg 2008) pp. 154–168

47.102 P:L. Lanzi: xcslib – The XCS Library. http://xcslib.sourceforge.net/

47.103 P. O. Stalph, M. V. Butz: Documentation of JavaXCSF (COBOSLAB, University of Würzburg, Germany, Y2009N001 2009)