State of XCS Classifier System Research

Stewart W. Wilson

The University of Illinois, Urbana-Champaign IL 61801, USA
Prediction Dynamics, Concord MA 01742, USA
wilson@prediction-dynamics.com

Abstract. XCS is a new kind of learning classifier system that differs from the traditional kind primarily in its definition of classifier fitness and its relation to contemporary reinforcement learning. Advantages of XCS include improved performance and an ability to form accurate maximal generalizations. This paper reviews recent research on XCS with respect to representation, internal state, predictive modeling, noise, and underlying theory and technique. A notation for environmental regularities is introduced.

1 Introduction

A classifier system is a learning system that seeks to gain reinforcement from its environment based on an evolving set of condition-action rules called classifiers. Via a Darwinian process, classifiers useful in gaining reinforcement are selected and propagate over those less useful, leading to increasing system performance. The classifier system idea is due to Holland [4], who laid out a framework that included generalizability of classifier conditions, internal message-passing and reinforcement, and computational completeness. However, despite considerable research, the performance of the traditional system has been mixed, and there have been few advances on the initial theory.

Recently, Wilson [30] introduced XCS, a classifier system that retains essential aspects of Holland's model, while improving on it in some respects. XCS's definition of classifier fitness is different from the traditional one, and the system's organization includes ideas from the contemporary field of reinforcement learning [25]. In particular, due to the fitness change—which is theoretically based—XCS tends to evolve classifiers that are both accurate in their prediction of payoff and maximally general, resulting in higher performance as well as compactness of representation. The traditional model provides no theory as to how generalization would occur accurately, and in fact it does not. XCS's performance gets further theoretical transparency from the reinforcement learning (RL) connection, particularly the use of predictions and discounting. Theoretical understanding of performance in the traditional model is limited.

XCS has been investigated on problems including learning the Boolean multiplexer functions, and to find goals in grid-like "woods" and maze environments. XCS reaches optimal performance in these problems, which are generally more difficult than problems set for the traditional system, where performance is rarely optimal.

Because XCS appears both theoretically and practically to be a clear advance, and also because these developments have come rather quickly, it is desirable to undertake a review of the current state of XCS research. The following discussion assumes a familiarity with classifier systems, as well as acquaintance with the basics of XCS [30]. The material is organized under five topics: representation, internal state, predictive modeling, noise and uncertainty, and XCS theory and technique. First, though, it will be useful to introduce some simple notation to describe classifier system environments.

2 Environment notation

A classifier system acts to gain payoff in its environment. Suppose that x represents a particular environmental state as detected by the system's sensors at time t and that a represents an action the system is capable of taking in that state. Let p represent an amount of payoff. Then the notation $(x, a) \rightarrow p$ says that if the system takes action a in state x, payoff p will be received. The expression states what we may term a *predictive regularity* of the environment.

With some further notation, we can represent the totality of predictive regularities. Let X stand for the set of all environmental states (as read from its sensors) that the system will encounter, let A represent its set of available actions, and let P represent the set of possible payoffs. Then there is a *mapping* $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ from the product set of states and actions to the set of payoffs, that expresses the totality of predictive regularities in the system's environment. The mapping can also be regarded as representing the system's *payoff* landscape, since it associates a value from P with every point in the space of sensor variables and actions defined by the $\mathrm{X} \times \mathrm{A}$ product set. (Not all points in the space may be realizable by the system).

Suppose that, for a given action a and payoff p, more than one state x_i satisfies $(x_i, a) \rightarrow p$. Let us write $(\{x\}, a) \rightarrow p$ to express all such states compactly, where we have dropped the subscript. We term the set $\{x\}$ a *categorical regularity*. It denotes a maximal set of states which are equivalent in the sense that they all satisfy $(x, a) \rightarrow p$ for a given value of a and a given value of p.

We can now look at the payoff landscape $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ from a categorical perspective. In general, the landscape will not be indefinitely "irregular". Instead, it will contain regions $(\{x\}, a)$ over which the value of p is constant, or nearly so. Each such region is a categorical regularity. The points within the region are equivalent with respect to payoff.

To summarize, a predictive regularity says that action a in state x leads reliably to payoff p. A categorical regularity says that, given a, there may be many states x which result in the same p.

Note, as an important aside, that often the payoff from the environment is *zero* for most pairs (x, a). This is the case in environments where the system must perform a sequence of actions in order to arrive in a state where payoff can be obtained. Thus, much of the time, the system must "find its way" without benefit of payoff directly from the environment, i.e., without so-called *external payoff* or reward. To deal with this situation, Holland proposed the *bucket brigade* algorithm, in which external payoff is in effect passed back from each activated classifier to its predecessor. The bucket brigade was in fact an early proposal for reinforcement learning in sequential environments.

The technique used in XCS is similar in some respects, but is closer conceptually and algorithmically to *Q-learning* [27], a theoretically grounded and widely used technique in contemporary reinforcement learning. The result is that XCS exists in an environment where the payoff for each (x, a) is a combination of external payoff (if there is any) and the system's current estimate of the maximum payoff available in the succeeding state y. Thus the payoff environment for XCS is a combination of payoffs from the "true" external environment and an internal estimation process modeled after Q-learning. With this qualification, we shall continue to characterize the environment by the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, at least until the next qualification!

3 Representation

In general, the objective of XCS is to capture, in the form of classifiers, the predictive regularities of the environment, $\mathcal{E}$, so the system can attain payoff at optimal rates. A second objective is to evolve a classifier population that captures $\mathcal{E}$'s categorical regularities compactly and accurately. In principle, compactness of representation can occur because classifier syntax permits the formation of *generalizations*, classifiers whose conditions match more than one state. In the traditional syntax, where the condition is a string from $\{1, 0, \#\}$, a condition can match more than one state if it contains #'s.

Ideally, for each categorical regularity $(\{x\}, a) \to p$ in $\mathcal{E}$, the system should evolve a classifier $\langle condition \rangle : \langle action \rangle \Rightarrow \langle prediction \rangle$ in which the action and prediction are a and p, respectively, and the condition exactly corresponds to $\{x\}$; that is, all and only the states in $\{x\}$ are matched by the condition. Put another way, the generalization expressed by the classifier should correspond exactly to the categorical regularity. Evolving such classifiers is important because it minimizes population size in terms of distinct classifiers (or *macroclassifiers*—see Section 7.4), resulting in rapid matching and visibility of the system's knowledge to human beings. Equally important, in many problems the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ cannot be represented by a reasonable number of classifiers *unless* its categorical regularities are captured. Finally, if classifiers are capable of expressing generalizations, it is important that the generalizations be *accurate*, i.e., that all of the state-action combinations expressed by the classifier indeed result in the same payoff. Otherwise performance will usually be less than optimal, since some classifier predictions will be incorrect, causing the system to choose suboptimal actions.

In light of these objectives, there are two broad respects in which the traditional syntax appears limited. First, the individual variables of the condition can take just two values, 1 and 0 (plus #). This is not a limitation where the environment's variables are also strictly two-valued. But in many environments, the variables of interest are continuous, or may take on more than two discrete values. Clearly, if $\mathcal{E}$ has continuous variables, it will only be in fortuitous cases that its categorical regularities can be accurately captured by thresholding to fit the {1,0,#} coding, and then only with just the right thresholds.

This problem can be alleviated to some extent by allowing variables in the condition to be expressed by binary numbers greater than 1, e.g., by four bits to represent a range from 0 to 15. The problem here is that many possible regularities of $\mathcal{E}$ are still not expressible. For example, 111# will match the range 14-15, but a range such as 13-14 cannot be expressed as a four-position string from {1,0,#}.

One solution [30, 32] would be to explore condition encodings where each variable is expressed by a real interval. For instance, the interval ($13.0 \leq v < 14.0$) would be expressed by the numbers 13.0 and 14.0 as an ordered pair, or, alternatively, by 13.5 as a center value and 0.5 as a "spread" or delta value around it. The classifier condition would be a concatenation of such pairs. The condition would be satisfied, or match, if each component of the input, a real vector, fell within the interval denoted by the corresponding ordered pair. The condition would thus express a conjunct of interval predicates. The interval predicates would be restricted to connected intervals (two or more separated intervals could not be expressed in one classifier), but that will in many cases be satisfactory. Classifiers with interval predicates would be evolved using genetic operators appropriate to real-valued variables. The application to practical areas like "data mining" is clear, since the variables there are often continuous.

The second limitation of the traditional syntax stems from the fact that the condition is a conjunct of individual predicates, i.e., an AND. For problems, such as Boolean functions, where the categorical regularities are expressible as ANDs of the variables, the syntax is appropriate because it matches $\mathcal{E}$'s own "syntax" directly. However, in other environments, the categorical regularities can be quite different, and therefore so is the optimal syntax.

For example, consider a robot having a linear array of light sensors around its periphery. Appropriate actions may depend on how much total stimulus is coming from one direction vs. another. A natural kind of classifier condition might compare the sum of certain sensor values with the sum of certain other values; if the first sum was greater, then a certain payoff would be predicted for an action like rotating to the left. Such a condition is impossible to express with the traditional syntax, and in fact would require a large number of separate classifiers. Instead, one would like a condition syntax that could directly express a predicate involving comparison of sums. Such a predicate would represent a categorical regularity with respect to all sensor value combinations that make it true. More generally, one would like a syntax permitting expression of *any* predicate, or truth-function of the input variables.

Predicates of any type can be constructed as Lisp s-expressions of primitive functions appropriate to the given environment. For example, in the problem just mentioned, appropriate primitives would include '+' and '>'. Genetic programming [10] has shown in principle how such predicates might be evolved, leading to the suggestion [29] of *s-classifiers*, classifiers with s-expression conditions. Recently, Lanzi and Perrucci [17] reported learning of the Boolean 6-multiplexer using XCS with s-classifiers. Learning times in terms of number of examples seen were approximately equal to times with the traditional syntax, suggesting—perhaps surprisingly—that use of s-expressions may not bring greater learning complexity. Their experiments with sequential problems in a "woods" environment were equally promising in this respect. The results encourage extension of XCS and s-classifiers to environments with a wide range of appropriate predicate functions.

It is worth noting that genetic programming and XCS use similar fitness measures. While in GP the fitness measure can vary with the problem domain, very often it involves the degree of fit between the candidate function and a number of pre-specified "fitness cases". That is, the fitness of the candidate is measured by the accuracy with which its output matches the values of the fitness cases. In XCS, the fitness of a classifier is measured by the accuracy with which it predicts the payoff that is received when it matches the input and the system chooses its action. The classifier's "fitness cases", so to speak, are the states it matches and the associated payoffs (for its action). This similarity suggests that much of the technique developed for GP should carry over well to XCS. It also suggests thinking of the classifier's condition primarily as a payoff-predicting function instead of as a predicate testing for a match. It is in fact reasonable to consider extensions in which the classifier predicts not a constant payoff value, but a value that varies with input state. The fitness of such a classifier would still be measured by the accuracy of the predicted values. Evolution of such classifiers could lead to further reductions in population size, since each could cover (x, a) pairs having different payoff values. In the limit, the population might evolve to consist of just one species of classifier, predicting all payoffs. This speculative outcome would be like the result of standard genetic programming, in which just one best individual is sought, but would occur by a different and possibly faster route.

S-classifier conditions are variable-length and, unlike traditional conditions, not all input variables need be represented. Missing variables are in effect "don't cares". The simplest kind of variable-length condition would concatenate the variables that are included as an unordered string of $\langle variable, value\rangle$ pairs. The format is the same as that employed by Goldberg, Korb and Deb [3] in the messy genetic algorithm. Lanzi [13] experimented with this condition format using genetic operators and other techniques similar to those in the messy GA, while leaving unchanged the rest of XCS. After encountering and solving certain problems of under- and over-specification, Lanzi was able to achieve optimal performance on problems also solved optimally by regular XCS. Lanzi's *messy classifier system* has the property that a population evolved for a certain set of sensors can be installed in a system having additional sensors and through mutation alone will continue evolving to take advantage of the additional sensors.

This follows from the position-independence and lack of fixed format of the messy CS, and is experimentally demonstrated in the paper. Lanzi suggests that position-independence "can improve the portability of the behaviors learned between different agents", a property that would presumably also hold for the s-classifier system.

The work on representation in XCS is aimed at increasing its learning power and ability to deal with payoff landscapes of any reasonable sort. Each payoff landscape has regularities. For high performance and representational efficiency, XCS must not only detect the regularities, but must also be able to represent them accurately. Pilot success with s-classifiers hints that this will be possible for any environment, but there is great room for further advances and understanding.

4 Predictive Modeling

XCS learns the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, a payoff model of $\mathcal{E}$, but the environment offers additional information that XCS could usefully predict. Specifically, given a state x and an action a, there results not only a payoff p, but a next state y. If XCS were to learn the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{Y}$, with elements $(x, a) \rightarrow y$, it could, from any present state x, consider chains of actions extending into the future; that is, it could plan. A representation of the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{Y}$ is usually termed a *predictive model.*

Holland [5] and Riolo [20] proposed classifier systems in which each classifier has a condition, an action, and a prediction of the resulting next state. The systems (which differed somewhat) were to learn predictive models of $\mathcal{E}$, but because the experiments did not involve the discovery or GA component, the systems's workability in practice is unknown. Sutton's [24] *Dyna* system used a predictive model coupled with standard reinforcement learning algorithms to speed up learning in goal-finding problems. Dyna learned the payoff mapping $\mathrm{X}\times\mathrm{A} \Rightarrow \mathrm{P}$ not only by trial and error in the actual environment, but also through hypothetical actions carried out in the predictive model as it was being built up, resulting in much faster learning of the payoff map, at least in terms of real actions in $\mathcal{E}$. Sutton experimented in environments with uniquely labeled states, and used tabular learning algorithms such as tabular Q-learning, so that no generalization over states occurred. However, the Dyna concept would carry over directly to XCS. Stolzmann's [23] Anticipatory Classifier System (ACS) predicts the next state and has been demonstrated experimentally in small problems. ACS does not use the GA, but instead generates a new, correct, classifier when existing classifiers fail to predict correctly, i.e., are inaccurate. The system has many interesting features that deserve further investigation.

An XCS classifier suitable for both payoff and predictive modeling could have the form $\langle condition \rangle : \langle action \rangle \Rightarrow \langle prediction \rangle : \langle expecton \rangle$, in which $\langle expecton \rangle$ is a vector of variables describing the succeeding state [30]. The fitness of this classifier, as in regular XCS, would be measured by the accuracy of the prediction of both payoff and the next state. Questions arise, of course. How is the accuracy of the expecton measured? Suppose $(x, a) \rightarrow p$ in $\mathcal{E}$, but the next state, y, is not definitely predictable; what form should the corresponding classifier have? To address these questions, suppose that the expecton can contain unspecified positions, along the lines of don't cares but in this case more like "don't knows". Then the expecton's accuracy would be measured by a function of the number of specified positions and the agreement between the values in those positions and the actual next state's values. In situations where the next state was not completely predictable, but some state variables were, a high fitness classifier could evolve which successfully predicted just those variables and left the others as "don't knows". The expecton would in effect generalize over next states.

It is not clear, however, that the payoff and predictive mappings should be learned by the same set of classifiers. Perhaps it is better to have classifiers of the form $\langle condition \rangle : \langle action \rangle \Rightarrow \langle prediction \rangle$ and of the form $\langle condition \rangle : \langle action \rangle \Rightarrow \langle expecton \rangle$. Then the underlying state regularities could be captured independently of the payoff configuration, which might change. Too, it could be that the payoff map was simple compared with the state map, and having separate classifier populations would exploit this. If the environment contained several kinds of payoff, with distinct payoff maps, use of a separate state classifier system would mean the state information had only to be learned once.

The best approaches to adding predictive modeling to XCS are completely open at present. The benefits include faster learning along the lines of Dyna, but incorporating generalization. Other benefits may stem from the fact that the state classifier system, if it involves significant generalization in its conditions and expectons, would yield a sort of simplified recoding of $\mathcal{E}$, that is, a simplified description of the state landscape. How this might be useful is not clear at present.

5 Internal state

At this point we must disturb the neat $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ environment model by introducing the Markov/non-Markov distinction. An environment has the *Markov property* if the prediction of p given x and a cannot be improved by knowledge of states preceding x. Strictly speaking, this is the Markov property with respect to payoff. Such an environment might or might not be Markov with respect to next states. But, for simplicity, by Markov we shall mean Markov with respect to payoff. An environment is *non-Markov* if it is not Markov.

The Markov/non-Markov distinction is crucial in reinforcement learning because it says whether an environment can or cannot be predicted on the basis of current input information. If so, the system can rely entirely on that information. If not, it must remember, or encode, something about the past and use that in conjunction with current input information in order to make correct decisions. If $\mathcal{E}$ is Markov (and deterministic), its payoff properties can be represented by $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$. If $\mathcal{E}$ is non-Markov, the mapping is not well defined because the p resulting from (x, a) is not predictable.

A very simple example of the distinction is the following [19]. Imagine a gift packing task in which a gift must be placed in an initially closed box after which the box is wrapped. Initially the box is closed with no gift inside and must be opened and filled. Later, with gift inside and again closed, it must be wrapped. In both cases, the system's sensors see a closed box, but the actions called for are different. If the system only sees the box, it can't perform correctly unless it remembers what it has done, or has made some other mental note, i.e., unless it maintains some form of internal state. The environment is non-Markov. If, however, the system's sensors include one that weighs the box, then for a high enough gift to box weight ratio, the environment becomes Markov and no internal state is required.

Note how the distinction very much depends on the system's sensors. Environments can be "made" Markov if there are enough sensors. Markov environments will become non-Markov if the number of sensors (or their ability to make distinctions) is reduced. Thus it is not the environment itself that is Markov or non-Markov, but the environment as sensed by the system. In speaking of an environment as Markov or non-Markov, we shall mean the environment in conjunction with the system's sensors.

How can XCS learn non-Markov environments? In reinforcement learning, there are two major approaches. One, the "history window", extends the input vector to include inputs from previous time-steps, enlarging the effective state space until the mapping $\mathrm{X}' \times \mathrm{A} \Rightarrow \mathrm{P}$, with X' the extended space, becomes well-defined, and the problem is effectively Markov. Though X' grows exponentially with the length of the history window, this approach can be satisfactory if the window is very short. The other approach attempts to identify and remember past events that will disambiguate $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, and these are combined with the current x to make decisions. Some versions of this approach employ recurrent neural networks. The problem here is that the complexity of the search is again exponential with distance into the past.

Holland's classifier system contains an internal message list, where information from one time-step can be posted by a classifier and then be read by another classifier on a later time-step. It was hoped that classifiers would evolve whose postings and readings led to high performance in situations where past information was needed (i.e, in non-Markov environments). Holland's idea was neither a history-window approach, nor one that searched purposively for critical past information. Rather, the approach was Darwinian, relying on variation and selection. Unfortunately, because of the complexity of the message list structure, and also—in our opinion—because fitness was based on "strength" and not accuracy as in XCS, Holland's system showed only limited success on non-Markov problems [21, 22].

Wilson [29] suggested that coupling between posting and reading classifiers would be enhanced if the internal state were embodied in a simple bit register instead of a list of messages. He observed that in many problems, decisions needing past information often require only one or a few bits. Classifier structure could take the form, $\langle \textit{external condition} \rangle \langle \textit{internal condition} \rangle$:

$\langle$*external action*$\rangle$ $\langle$*internal action*$\rangle \Rightarrow \langle$*prediction*$\rangle$. The internal condition would apply to the internal bit register; the internal action would set, reset, or ignore bits there.

The suggestion was taken up by Cliff and Ross [2]. They added an internal register to Wilson's [29] ZCS system (a classifier system with fitness based traditionally on strength). The resulting system, ZCSM, performed quite well, though not optimally, in several simple non-Markov mazes. Examination of evolved populations showed that ZCSM indeed evolved classifiers that employed the internal register to discriminate the mazes' "aliased" (indistinguishable) states.

Lanzi [11] obtained improved results with XCSM, which added an internal register to XCS, but more difficult mazes could still not be learned optimally. In further work [12], he proposed that exploration of internal actions should be carried out differently than exploration of external actions. He found that too much exploration of internal actions would prevent the stable formation of classifiers that acted appropriately on the internal register contents. To use the register effectively implies that the system must both create an appropriate internal "language" (of register settings) and also learn how to interpret that language. Lanzi experimented with XCSMH, a modified XCSM in which the internal register settings (the "language") were explored only by the genetic algorithm and were not explored as part of the action selection process. The language was thus explored more slowly than its interpretation. This resulted in optimal performance on more difficult (more aliased states) mazes. In further work [14], he discovered that still improved performance would occur if the internal register was somewhat redundant, that is, had size somewhat greater than the minimum number of bits needed in principle to disambiguate the environment's aliased states. This last result confirms, in classifier systems, a similar result observed by Teller [26] who used an internal register with genetic programming.

Cliff and Ross [2] worried that systems using an internal register would not scale up well, because the amount of needed exploration would grow exponentially with the register size. Lanzi's results suggest this may not be a concern, since better performance results from less internal exploration than might have been thought. In effect, it is not necessary to explore all possible settings of the register. Many settings will be interpretable to yield high or optimal performance; it is only necessary to find one of them. A broad discussion of the non-Markov work is given in [18].

Good learning in non-Markov environments is perhaps the largest outstanding problem in RL. Progress using an internal register with XCS suggests this approach should be pursued vigorously. Most important would seem to be to study further how best to do "internal exploration", to understand better the complexity implications of the register and its size, and of course to investigate increasingly difficult environments. Learning of hierarchical behavior is another outstanding RL problem. It is possible that internal "languages" will evolve that use the register in a hierarchical manner, i.e., certain bit positions will encode longer-term contexts while others encode behavioral details. This might open the way to realization of "hierarchical classifier systems" [28].

6 Noise and uncertainty

The vast majority of classifier system research has used noiseless, deterministic environments, but these ideal properties do not hold in many potential applications. Robotic sensors are often extremely noisy, as are the data in most "data mining" problems. Besides noise inherent in the environment, XCS is vulnerable to noise-like uncertainty in two other respects. First, if the environment is non-Markov and the system has insufficient internal state to disambiguate it, the system's classifiers will not be able to arrive at stable predictions. Classifier errors will not go to small values or zero and this error will be indistinguishable from error that results from "true" environmental noise. Second, if a classifier is overgeneral, it will make prediction errors that are also not distinguishable from the other kinds, at least until the overgenerality is eliminated.

Since in XCS fitness is based on accuracy, it is important to determine XCS's sensitivity to external noise. Lanzi and Colombetti [16] tested XCS in Markov environments where the system's probability of carrying out its intended action was less than 1.0. Specifically, in a grid-world maze, the system would with probability ϵ "slip" to one or the other of the two cells adjacent to its intended destination cell. Thus the system was subject to a kind of action noise. For values of ϵ up to about 0.25, Lanzi and Colombetti found that performance was nearly as good as XCS without generalization (no #'s) or tabular Q-learning, all subject to the same action noise. Thus even with generalization "turned on", XCS was able evolve classifiers as accurate as those evolved when generalization was not enabled. The system was able to eliminate errors due to overgeneralization even in the presence of external noise over which it had no control. However, at an ϵ of 0.50, performance broke down drastically.

The authors then introduced a technique that resulted in nearly optimal performance even when ϵ was 0.50. A new parameter μ was given each classifier for keeping an estimate of the minimum prediction error among the classifiers in its action set [A]. Each time a classifier takes part in [A], it looks at the other classifiers in [A] and notes the value of the prediction error parameter of the classifier with the lowest such parameter. Our original classifier then updates its μ parameter using the value just identified. Finally, when it updates its own prediction error parameter, it uses not its current prediction error, but that error minus its current value of μ. The procedure is based on the heuristic that the minimum prediction error in the action set is a good estimate of the actual external noise, since all the classifiers are subject to it. Subtracting μ out means that each classifier's error estimate approaches the value it would have in a noiseless environment, so that overgeneralization errors can be eliminated and performance is maintained. The technique is a new kind of example of how the population-based nature of XCS—the fact that it supports multiple hypotheses for each situation—can markedly aid the search.

It is important to test the technique on the other kinds of environmental noise, namely noise in the sensors and in external payoffs. Both are important for prediction from real data, since data sets used for learning will often contain errors in data (sensor) values and inconsistencies in outcomes (payoffs). Lanzi and Colombetti's technique promises to be valuable in learning predictions from noisy data while retaining the ability to detect generalizations, properties which will be of great interest to human users of XCS in data applications.

7 XCS theory and technique

7.1 Generalization

From the beginning, XCS has displayed a strong tendency to evolve classifiers that detect and accurately represent the categorical regularities of its environment. This is based, first, on making fitness depend on accuracy of prediction (in contrast to "strength" in the Holland framework). Second, it depends on the use of a niche GA [1]. Together, these have the consequence that of two equally accurate classifiers where one matches a subset of the states matched by the other, the more general classifier will win out because it has more reproductive opportunities (for a detailed discussion, see [30]).

The drive to evolve accurate, maximally general classifiers at the same time as it learns the mapping $X \times A \Rightarrow P$ suggests that XCS tends to evolve populations that consist of the smallest possible set of non-overlapping classifiers, thus representing the environment's payoff landscape optimally. Kovacs [7] termed this the *XCS Optimality Hypothesis*, and demonstrated it for Boolean multiplexer problems. It is important to explore this direction further, in particular by trying Boolean problems which are less symmetrical than the multiplexers. In the multiplexer function, each categorical regularity covers an equal portion of the input domain. As a result, for random inputs, all parts of the population are updated and subjected to GA processes at approximately the same rate. For a function with categorical regularities of unequal size, this equality of rates would not hold and one can expect that XCS's functionality will be affected to a degree that should be determined. A similar kind of test would be achieved with the multiplexer by changing to an input distribution that was not uniformly random.

XCS's tendency to evolve accurate, maximally general classifiers is not guaranteed. Situations can arise, particularly in sequential problems using discounting, in which overgeneral classifiers may fail to be eliminated, even though their accuracy is low. Elimination depends on the existence of a more accurate competitor classifier in every action set where the overgeneral occurs. Normally this will be the case, due to the genetic mechanisms. However, if populations are too small relative to the number of distinct categorical regularities in $\mathcal{E}$, action set sizes will be small so that the GA will be slow to generate the competitors and an overgeneral may not be eliminated before it causes trouble, i.e., before action errors occur that in the worst case bring a breakdown of overall performance.

Overgenerals will also not be eliminated if the errors in an action set are so small that the system's accuracy mechanism cannot differentiate the fitnesses. Discounting can have this effect, since the predictions—and thus errors—become smaller farther from sources of external payoff. XCS originally defined accuracy as a negative exponential function of prediction error. If one very small error is nevertheless twice as big as another one in the same action set, an exponential function will not differentiate them well since the ratio of two exponentials is an exponential of the difference of their arguments, and may be small. The problem is substantially reduced by changing the accuracy function to a negative power function of the error [31]. Then, if one error is twice another, the accuracy ratio is a power of two, independent of the actual error values.

Despite larger populations and an improved accuracy function, performance can still sometimes break down due to failure to eliminate overgenerals rapidly enough. For such "emergencies", Lanzi [15] developed a mechanism termed *specify*. Every action set is tested to see if its average error relative to the population average error exceeds a threshold. If so, a new classifier is added to the action set that covers the current input and whose condition has a predetermined probability of don't care positions. The intention is to add a fairly specific classifier to the high-error action set that will compete in accuracy with the overgeneral classifiers that are probably present. Specify works well in that performance breakdowns due to overgeneralization are eliminated. At the same time, because it acts locally, specify does not restrain the system from forming accurate generalizations where warranted.

In the first work on XCS, the GA occurred in the match set [M]. Later [31] it was moved to the action set [A]. The result was that in certain problems the population size—in terms of macroclassifiers—evolved to be smaller. The smaller populations contained the same accurate general classifiers as before, but there were fewer more-specific versions of these present, and fewer inaccurate classifiers. The apparent reason for this improvement is fairly subtle.

Consider a match set, and suppose that the state x being matched participates in two categorical regularities of $\mathcal{E}$: $(\{x\}_1, a_1) \rightarrow p_1$ and $(\{x\}_2, a_2) \rightarrow p_2$. Suppose classifiers C1 and C2 in [M] perfectly represent these regularities. That is, the condition of C1 exactly fits $\{x\}_1$ and the condition of C2 exactly fits $\{x\}_2$. Both classifiers are accurate and maximally general. What if the GA crosses them? If the two conditions are identical, then the offspring conditions will be identical to the parent conditions. However, if the two conditions are *not* identical, the offspring conditions may be different from those of both of the parents (e.g., cross #1 and 1#). Since by hypothesis the parents were accurate and maximally general, the offspring, if different from the parents, will not be and will thus constitute a kind of crossover "noise" that the system will have to eliminate. Thus even if the system has found accurate, maximally general classifiers that match the present input, the action of crossover will—if $\{x\}_1$ and $\{x\}_2$ differ—continue to generate suboptimal classifiers.

Suppose however that the GA, and thus crossover, occurs in [A] instead of [M]. Now only one of C1 and C2 is present and not the other (they have different actions). Since they cannot be crossed, the offspring just mentioned will not be formed and do not have to be eliminated. C1 (suppose it's the one present) will still cross with more-specific and more-general versions of itself that may be present too in the action set; the same crosses would occur in the match set.

But the "noise" due to crosses with C2 will not. The argument is clearest in an extreme case: assume the match set consists *only* of C1 and C2.

The reduction in population size due to moving the GA to [A] was observed in problems where, if a classifier was accurate and maximally general, it was often not the case that another classifier with a different action but the same condition would also be accurate and maximally general (another way of saying that C1's condition is different from C2's). However, in problems where this was the case (conditions of C1 and C2 always the same), the population size was unchanged in moving the GA from [M] to [A]. Thus the shift to the action set appears justified both in principle and experimentally. But further examination in both respects is needed to understand fully what is happening.

If the GA does occur in the action set, what can we say that crossover is actually doing? Note that all classifiers in both the match and action sets must match the input. However, all classifiers in the action set have the same action. Thus the GA may be regarded as searching for the best—i.e. most general and accurate—classifier to represent the regularity $(\{x\}, a) \rightarrow p$ that the present (x, a) pair belongs to. The classifiers in the action set have conditions of varying specificity. They exist somewhere along the line between completely specific and completely general. Crossover produces offspring that in general occupy points on the line different from their parents. Clearly the GA—using crossover, mutation, and selection—is searching along this line, driven by a fitness measure, accuracy, that is strongly correlated with specificity. This is the heart of XCS's discovery process, and a detailed theory is definitely called for.

Also called for is a theory of how fitness based on accuracy interacts with the reproductive opportunity afforded by greater generality to drive the system toward classifiers that are both accurate and maximally general. Wilson [30] presented a heuristic argument: of two equally accurate classifiers, the more general one would win out because being more general it occurs in more action sets and thus has greater chance to reproduce. But the theory needs to be put on a quantitative basis.

Wilson [31] reported a secondary generalization method, *subsumption deletion*, that further reduces, or "condenses", population size. The primary method, as discussed above, causes classifiers to generalize up to the point where further generalization (e.g., addition of #'s in the traditional syntax) would result in errors. However, the process may actually stop short of this point, since a formally more general classifier (more #'s) will only win out if it can match more states of the environment (yielding more reproductive opportunities). But those states may not actually be present in $\mathcal{E}$, i.e., $\mathcal{E}$ may not contain all the states permitted by the encoding. So the system will not evolve classifiers that are as formally general as they could in fact be.

To see this, suppose $\mathcal{E}$ contains the categorical regularity $(\{000, 001\}, a) \rightarrow p_1$. Suppose $\mathcal{E}$ also contains $(100, a) \rightarrow p_2$, where p_2 and p_1 are different. Suppose further that the states 010 and 011 are not present. The classifier $00\# : a \Rightarrow p_1$ is clearly accurate. So also is $0\#\# : a \Rightarrow p_1$. The second classifier is more general, but since it matches no more actual states than the first, it will not win out reproductively, and the two will coexist in the population. Note, however, that the classifier $\#\#\# : a \Rightarrow p_1$, being inaccurate, will not survive. (See also the discussion in [15]).

In many problems, it is desirable to end up with classifiers that are as formally general as possible while still being accurate. The resulting population is smaller, and the key differentiating variables (e.g., the first bit in the example above) are more perspicuous. Subsumption deletion accomplishes this by checking, whenever a new classifier is generated by the GA, whether its condition is logically subsumed by the condition of an accurate classifier already in the action set. If so, the new classifier is abandoned and not added to the population. Subsumption deletion is a powerful addition to the primary generalization mechanism, but it is somewhat risky. For each categorical regularity, it tends to eliminate all but the formally most general, accurate classifier. If the environment should change, such that certain states (e.g., 010 and 011 above) now occur, that classifier could fail drastically resulting in a breakdown of performance. Retaining classifiers like $00\# : a \Rightarrow p_1$ above—i.e., using only the primary process— prevents this.

7.2 Connection with RL

XCS's learning algorithm is a straightforward adaptation of basic Q-learning. The prediction of each classifier in the action set is updated by the current reward (if any) plus the discounted value of the maximum system prediction on the next time-step. The system prediction for a particular action is a fitness-weighted average of the predictions of each classifier in the match set that advocates that action. The maximum system prediction is the maximum, over the match set, of the system prediction for each action. The essential difference with Q-learning is that classifier—that is, rule—predictions are updated from predictions of other rules. In Q-learning, action-values—that is, predictions for pairs (x, a)—are updated from other action-values. XCS's memory is contained in rule sets, whereas Q-learning's memory is stored in a table with one entry for each state-action pair.

Of course, Q-learning-like algorithms have also been employed, for example, in neural-net learning. But it is most interesting to compare XCS with tabular Q-learning. That is the only case for which convergence proofs are known. In addition, there is an important sense in which XCS's algorithm is like tabular Q-learning, but with generalization over the entries of the table.

If XCS is applied to a problem, but with generalization turned off (no initial #'s, and none introduced by mutation), the result will be a set of classifiers corresponding to the equivalent Q-learning table, except that for most state-action pairs that do not actually occur in $\mathcal{E}$ there will be no classifier. Thus, without generalization, XCS will in effect build the Q-learning state-action table "from scratch", but only for (x,a) pairs that actually occur in $\mathcal{E}$. Performance, apart from some noise introduced by the GA, will be identical to that of tabular Q-learning. If the GA is completely turned off, and classifiers are only created by covering, performance will be the same as for tabular Q-learning, and the final population will correspond precisely to Q-table entries that actually occur in $\mathcal{E}$.

With generalization turned on, performance by XCS will normally reach the same level as for tabular Q-learning, but will take longer due to errors as accurate general classifiers are discovered and emphasized. If the problem contains categorical regularities and these can be represented by XCS's syntax, then generalization will result in a population (in macroclassifiers) that is smaller than the corresponding Q-table, often by a significant factor. If one examines classifier predictions once the system has essentially stopped evolving, they are found to equal the predictions of the corresponding Q-table. Thus while no proofs of convergence are available, empirically XCS converges like Q-learning, but with generalization as a bonus. The same sort of parallel might hold with other reinforcement learning algorithms, provided XCS's update procedures were correspondingly modified. (See [9, 14] for detailed discussion of the relation between XCS and Q-learning.)

7.3 Complexity

Real-world problems often have very large state spaces. Reinforcement learning methods such as Q-learning that depend on tables of values scale exponentially with the dimensionality of the state space. Not only does the table grow exponentially, but so does the time needed to fill in the values. Tabular Q-learning is not sensitive to categorical regularities in the environment and so cannot avoid the exponential explosion. But research with XCS has suggested that because it detects the regularities—and when it can represent them syntactically—XCS's learning complexity is dependent not on the dimensionality of the space, but on the complexity of the underlying regularities. In this respect it may differ significantly from other approaches to handling large state-spaces such as radial basis functions, tiling (CMAC) methods, and neural networks, whose complexities are ultimately tied to state-space size [25].

Information about XCS's learning complexity comes from experiments with the family of Boolean multiplexer functions [31]. Three functions were learned: the 6-, 11-, and 20-multiplexers, where the numbers indicate the lengths of the input bit-string l or in other words, the dimensionality of the state space. The disjunctive normal forms (DNF) for the functions contain, respectively, 4, 8, and 16 terms. Associated with each term are exactly four payoff regularities, so there are, respectively, 16, 32, and 64 payoff regularities in the three spaces. An example of a payoff regularity for the 6-multiplexer is
$(\{000000, 000001, 000010, 000011, 000100, 000101, 000110, 000111\}, 0) \Rightarrow p$,
where p is the payoff associated with a correct decision. The accurate, maximally general classifier corresponding to this regularity is $000\#\#\# : 0 \Rightarrow p$.

XCS reached 100% performance on the three functions after seeing, on average, 2,000, 10,000, and 50,000 random input strings. Final population sizes in macroclassifiers were, respectively, 55, 148, and 345. In contrast, the state space sizes are, respectively, 64, 2,048, and 1,048,576, growing exponentially. If one infers from these limited data that the learning times grow by a factor of five from one multiplexer to the next, then the times can be fit to a function cg^p, where g is the number of regularities in the function, $p = \log 5 = 2.32$, and $c =$

3.22. Thus the learning time complexity would be a low power of the number of regularities in the space.

A very tentative theory of this result can be proposed. The search for a set of classifiers that accurately represent the regularities may involve two factors. One is the number of regularities itself. The other is the length of the classifier condition, because each position of the condition must be "set" correctly. The length of the condition is l, but the number of regularities is approximately proportional to l—it approaches an exact proportionality in the limit of large multiplexers. It seems reasonable to conclude that the search time should depend on product of these two factors, which would be consistent with the function just proposed.

Further research is needed to test these ideas. In particular, XCS should be applied to larger multiplexers, and to other function families. At this point, however, XCS promises to scale up well in problems for which its representational syntax is appropriate.

7.4 Techniques

Traditionally, classifier system populations contain many classifiers that, due to reproduction without associated crossover or mutation, are structurally identical, i.e., they have the same conditions and actions. Wilson [30] introduced *macroclassifiers* in order to eliminate this redundancy as well as reduce processing time and save storage. In a population of macroclassifiers, each newly generated classifier is examined to see if a structurally identical classifier already exists. If so, the existing classifier's numerosity parameter is increased by one and the new classifier is abandoned. If not, the new classifier is added to the population with its numerosity initialized at one. XCS uses a macroclassifier population, but all operations are otherwise conducted as though the population consists of ordinary classifiers ("microclassifiers"); that is, numerosities are taken into account. In a macroclassifier population, the sum of the numerosities, N, equals the number of underlying microclassifiers and is a constant. N is the number that in traditional systems denotes the population size.

A population of macroclassifiers consists entirely of structurally unique individuals. As such it permits a much better view of the system's "knowledge" than does a traditional population. Classifiers with high numerosity tend to be those with maximal accurate generalizations, so the most significant knowledge can be determined readily by sorting the population by numerosity [7]. Furthermore, the size of the population in macroclassifiers measures the system's ability to detect and represent the regularities in $\mathcal{E}$. The macroclassifier idea is in fact what permits realization of the improvement in space complexity that XCS achieves through generalization. A traditional population would maintain size N regardless of the generalizations within it.

Kovacs [6] compared the behavior of XCS with and without macroclassifiers on the 6-multiplexer problem. He found no significant difference in performance. A slight improvement in system prediction error using macroclassifiers was explainable since when a new classifier already exists it is not introduced into the population, so its arbitrary initial parameters can't affect system predictions. It therefore appears that the macroclassifier technique is valid and uniformly beneficial.

Whenever XCS generates a new classifier, either through the GA or covering, a classifier is effectively deleted in order to maintain the microclassifier population size at N. Selecting a classifier for deletion is done as though the population consists of microclassifiers. The actual "deletion" is carried out by simply decrementing some classifier's numerosity by one; if the classifier's numerosity is exactly one, it is removed from the population.

Kovacs [8] studied techniques for making the deletion selection and introduced a new one that is superior. Previously [30], two techniques had been used. Each classifier kept an estimate of the number of classifiers in the action sets it occurred in. In the first technique, the probability of deletion was proportional to the estimate. In the second technique, it was proportional to the estimate but multiplied by an integer greater than one if the classifier's fitness was below the population average fitness by a certain factor. Use of the action set size estimate tends to maintain action sets at about the same size, thus equally distributing system resources (classifiers). The integer multiplier in the second technique was designed to penalize very low fitness classifiers and eliminate them rapidly.

The second technique worked better than the first in that it resulted in smaller population sizes. But it often deleted newly generated classifiers before they had a chance to gain fitness. Kovacs's new technique resolved this by combining the two previous ones. Each classifier was given an additional parameter keeping track of the number of times it had been a member of an action set. In the Kovacs technique, if that number was less than, e.g., 20, the probability of deletion was as in the first old technique. If the number was equal to or greater than 20, the second old technique was used. The new technique exhibited the advantages of the earlier two without their disadvantages.

8 Conclusions

We have reviewed the current state of XCS classifier system research, identifying the main accomplishments and suggesting future directions. A notation for the environment's predictive and categorical regularities was introduced to aid the discussion.

Promising areas for further research include: (1) extending condition syntax so as to accept input vectors with continuous, ordinal, nominal, and mixed components, and to enable representation of a greater range of environmental regularities; (2) continued exploration of the full representational generality of s-classifiers; (3) experimentation with XCS systems that predict the next state, so as to develop more complete models of the environment; (4) continued work on internal state to determine the scope of the register method and its potential for supporting hierarchical behavior; (5) further research on filtering noise of all types, with applicability to learning from real data sets; (6) continued basic work on accurate generalization, aimed at understanding all relevant phenomena; (7) development of a schema theory for XCS's genetic search process; (8) exploration of RL techniques other than Q-learning in XCS; (9) further investigation of XCS's learning complexity experimentally and theoretically; and (10) continued examination of the basic XCS component algorithms in search of improvements.

Important areas not discussed because next to nothing is presently known include: (1) use of continuous actions, e.g., turn exactly 34 degrees; and (2) basing XCS on continuous time, instead of discrete time-steps. Existing RL work in both of these areas could be brought to bear. In addition, it is important to elucidate the relationship between XCS and traditional classifier systems [9].

XCS is a new kind of classifier system showing significant promise as a reinforcement learner and accurate generalizer. Its full potential is just beginning to be explored.

References

1. Booker, L.: Intelligent behavior as an adaptation to the task environment. Ph.D. Dissertation (Computer and Communication Sciences). The University of Michigan (1982).
2. Cliff, D. and Ross, S.: Adding temporary memory to ZCS. Adaptive Behavior **3**(2) (1994) 101-150.
3. Goldberg, D. E., Korb, B., and Deb, K.: Messy genetic algorithms: motivation, analysis, and first results. Complex Systems **3** (1989) 493-530.
4. Holland, J. H.: Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), Machine Learning, An Artificial Intelligence Approach. Volume II. Los Altos, California: Morgan Kaufmann (1986).
5. Holland, J. H.: Concerning the emergence of tag-mediated lookahead in classifier systems. Physica D, **41** (1990) 188-201.
6. Kovacs, T.: Evolving Optimal Populations with XCS Classifier Systems. MSc. Dissertation, Univ. of Birmingham, UK (1996).
7. Kovacs, T.: XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions. In: Roy, Chawdhry and Pant (Eds), Soft Computing in Engineering Design and Manufacturing, Springer-Verlag London (1997) 59-68.
8. Kovacs, T.: Deletion schemes for classifier systems. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 329-336.
9. Kovacs, T.: Strength or accuracy? A comparison of two approaches to fitness calculation in learning classifier systems. This volume.
10. Koza, J. R.: Genetic Programming. Cambridge, MA: The MIT Press/Bradford Books (1992).
11. Lanzi, P. L.: Adding memory to XCS. In: Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98). IEEE Press (1998).
12. Lanzi, P. L.: An analysis of the memory mechanism of XCSM In J. Koza et al (eds.), Proceedings of the Third Annual Genetic Programming Conference, San Francisco, CA: Morgan Kaufmann (1998) 643-651.

13. Lanzi, P. L.: Extending the representation of classifier conditions, part I: from binary to messy coding. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 337-344.
14. Lanzi, P. L.: Reinforcement Learning with Classifier Systems. PhD Thesis, Politecnico di Milano (1999).
15. Lanzi, P. L.: An analysis of generalization in the XCS classifier system. Evolutionary Computation, **7**(2) (1999) 125-149.
16. Lanzi, P. L. and Colombetti, M.: An extension to the XCS classifier system for stochastic environments. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 353-360.
17. Lanzi, P. L. and Perrucci, A.: Extending the representation of classifier conditions, part II: from messy coding to s-expressions. In: Banzhaf, W., et al, (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99). Morgan Kaufmann: San Francisco, CA (1999) 345-352.
18. Lanzi, P. L. and Wilson, S. W.: Toward optimal classifier system performance in non-Markov environments. Evolutionary Computation, to appear (2000).
19. Lin, L.-J.: Reinforcement Learning for Robots Using Neural Networks. Ph.D. Dissertation (School of Computer Science), Carnegie Mellon University (1993).
20. Riolo, R. L.: Lookahead planning and latent learning in a classifier system. In J.-A. Meyer and S. W. Wilson (eds.), From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press (1991) 316-326.
21. Robertson, G. G. and Riolo, R. L.: A tale of two classifier systems. Machine Learning, **3** (1988) 139-159.
22. Smith, R. E.: Default Hierarchy Formation and Memory Expoitation in Learning Classifier Systems. Ph. D. Dissertation, The University of Alabama, Tuscaloosa, Alabama (1991).
23. Stolzmann, W.: Anticipatory classifier systems. In: Proceedings of the Third Annual Genetic Programming Conference, J. Koza et al (eds.). San Francisco, CA: Morgan Kaufmann (1988) 658-664.
24. Sutton, R. S.: Reinforcement learning architectures for animats. In: J.-A. Meyer and S. W. Wilson (eds.), From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior. Cambridge, MA: MIT Press (1991) 288-296.
25. Sutton, R. S. and Barto A. G.: Reinforcement Learning: An Introduction. Cambridge, MA: The MIT Press/Bradford Books (1998).
26. Teller, A.: The evolution of mental models. In: K. E. Kinnear, Jr. (ed.), Advances in Genetic Programming. Cambridge, MA: The MIT Press/Bradford Books (1994).
27. Watkins, C. J. C. H.: Learning from Delayed Rewards. Ph.D. Dissertation, Cambridge University (1989).
28. Wilson, S. W.: Hierarchical credit allocation in a classifier system. In: Proceedings of the Tenth International Joint Conference on Artificial Intelligence. Los Altos, CA: Morgan Kaufmann (1987) 217-220.
29. Wilson, S. W.: ZCS: a zeroth level classifier system. Evolutionary Computation **2**(1) (1994) 1-18.
30. Wilson, S. W.: Classifier fitness based on accuracy. Evolutionary Computation **3**(2) (1995) 149-175.

31. Wilson, S. W.: Generalization in the XCS classifier system. In Proceedings of the Third Annual Genetic Programming Conference, J. Koza et al (eds.). San Francisco, CA: Morgan Kaufmann (1998) 665-674.
32. Wilson, S. W.: Get real! XCS with continuous-valued inputs. This volume.