4

The XCS Classifier System

The creation of the accuracy-based classifier system XCS (Wilson, 1995) can be considered a milestone in classifier system research. XCS addresses the general LCS challenges in a very distributed fashion. The problem of generalization is approached by niche reproduction in conjunction with panmictic (population-wide) deletion. The problem of strong overgenerals is solved by deriving classifier fitness from the estimated accuracy of reward predictions instead of from the reward predictions themselves. In effect, XCS is designed to not only evolve a representation of the best solution for all possible problem instances but rather to evolve a complete and accurate *payoff map* of all possible solutions for all possible problem instances.

This chapter introduces the XCS classifier system. We provide a concise description of problem representation and all fundamental mechanisms. The algorithmic description found in Appendix B provides an exact description of all mechanisms in XCS facilitating the implementation of the system. After the introduction of XCS, Section 4.2 shows XCS's performance on simple toy problems. Chapters 5 and 6 investigate how and when XCS is able to learn a solution developing a theory of XCS's learning capabilities. Subsequent chapters then evaluate XCS in more challenging problems, analyze learning and scalability, and introduce several improvements to the system.

4.1 System Introduction

As with most LCSs, XCS evolves a set of rules, the so-called *population* of *classifiers*, by the means of a steady-state GA. A classifier usually consists of a condition and an action part. The condition part specifies when the classifier is applicable and the action part specifies which action, or classification, to execute. XCS differs from other LCSs in its GA application and its fitness approach. This section gives a concise introduction to XCS starting with knowledge representation, progressing to learning iteration, and ending with learning evaluation and the genetic learning component. The introduction focuses on binary problem representations. Nominal and real-valued representations are introduced in Chapter 9.

4.1.1 Knowledge Representation

The population of classifiers represents a problem solution in a probabilistic disjunctive normal form where each classifier specifies one conjunctive term in the disjunction. Thus, each classifier can be regarded as an expert in its problem subspace specifying a confidence value about its expertise.

Each classifier consists of five main components and several additional estimates.

1. *Condition part* C specifies when the classifier is applicable
2. *Action part* A specifies the proposed action (or classification or solution)
3. *Reward Prediction* R estimates the average reward received when executing action A given condition C is satisfied
4. *Reward prediction error* ε estimates the mean absolute deviation of R with respect to the actual reward
5. *Fitness* F estimates the scaled, relative accuracy (scaled, inverse error) with respect to other, overlapping classifiers

As in LCS1, in the binary case condition part C is coded by $C \in \{0, 1, \#\}^L$ identifying a hyperrectangle in which the classifier is applicable, or *matches*. Action part $A \in \mathcal{A}$ defines one possible action or classification. Reward prediction $R \in \Re$ is iteratively updated resulting in a moving average measure of encountered reward received in the recent problem instances in which condition C matched and action A was executed. Similarly, the reward prediction error estimates the moving average of the absolute error of the reward prediction. Finally, fitness estimates the moving average of the accuracy of the classifier's reward prediction relative to other classifiers that were applicable at the same time.

Each classifier maintains several additional parameters. The *action set size estimate as* estimates the moving average of the action sets it is applied in. It is updated similar to the reward prediction R. The *time stamp ts* specifies the time when last the classifier was part of a GA competition. The *experience* counter *exp* counts the number of parameter updates the classifier underwent so far. The numerosity *num* specifies the number of identical micro-classifiers, this (macro-)classifier actually represents. In this way, multiple identical classifiers can be represented by one actual classifier in the population speeding up computation (for example, for the matching process).

4.1.2 Learning Interaction

Learning usually starts with an empty population. Alternatively, the population may be initialized generating random classifiers whose condition parts have an average specificity of $1 - P_\#$ (that is, each attribute in the condition part is a don't care symbol with probability $P_\#$ and zero or one otherwise).

Given current problem instance $s \in \mathcal{S}$ at iteration time t, the *match set* $[M]$ is formed, containing all classifiers in $[P]$ whose conditions match s. If some action is not represented in $[M]$, a covering mechanism is applied.[1] Covering creates classifiers that match s (inserting #-symbols, similar to when the population is initialized at random, with a probability of $P_\#$ at each position) and specify unrepresented actions. Given a match set, XCS can estimate the payoff for each possible action forming a *prediction array* $P(\mathcal{A})$,

$$P(A) = \frac{\sum_{cl.A=A \wedge cl \in [M]} cl.R \cdot cl.F}{\sum_{cl.A=A \wedge cl \in [M]} cl.F}. \tag{4.1}$$

We use the dot notation to refer to classifier parameters in XCS to avoid confusion with classifier unrelated parameters. Essentially, $P(A)$ reflects the fitness-weighted average of all reward prediction estimates of the classifiers in $[M]$ that advocate classification A. The prediction array is used to determine the appropriate classification. Several action selection policies (that is, behavioral policies) may be applied. Usually, XCS chooses actions randomly during learning, and it chooses the best action $A_{max} = \texttt{arg max}_A P(A)$ during testing. All classifiers in $[M]$ that specify the chosen action A comprise the action set $[A]$.

After the execution of the chosen action, feedback is received in the form of scalar reward $R \in \Re$, which is used to update classifier parameters. Finally, the next problem instance is received and the next problem iteration begins, increasing the iteration time t by one.

4.1.3 Rule Evaluation

XCS iteratively updates its population of classifiers with respect to the successive problem instances. Parameter updates are usually done in this order: prediction error, prediction, fitness.

In a classification problem, classifier parameters are updated with respect to the immediate feedback R in the current action set $[A]$. In an RL problem, all classifiers in $[A]$ are updated with respect to the immediate reward R plus the estimated discounted future reward as follows:

$$Q = max_{A \in \mathcal{A}} P^{t+1}(A), \tag{4.2}$$

where the $(t+1)$ term refers to the prediction array in the consequent learning iteration $t+1$.

Reward prediction error ε of each classifier in $[A]$ is updated by:

[1] Covering is sometimes controlled by the parameter θ_{mna} that requires that at least θ_{mna} actions are covered. For simplicity, we set θ_{mna} per default to the number of possible actions or classifications $|\mathcal{A}|$ in the current problem.

$$\varepsilon \leftarrow \varepsilon + \beta(|\rho - R| - \varepsilon), \tag{4.3}$$

where $\rho = r$ in classification problems and $\rho = r + \gamma Q$ in RL problems. Parameter $\beta \in [0, 1]$ denotes the learning rate influencing accuracy and adaptivity of the moving average reward prediction error. Similar to the learning rate dependence in RL (see Chapter 2), a higher learning rate β results in less history dependence and thus faster adaptivity but also in a higher variance if different reward values are received.

Next, reward prediction R of each classifier in $[A]$ is updated by:

$$R \leftarrow R + \beta(\rho - R), \tag{4.4}$$

with the same notation as in the update of ε. Note how XCS essentially applies a Q-learning update. However, Q-values are not approximated by a tabular entry but by a collection of rules expressed in the prediction array $P(\mathcal{A})$.

The fitness value of each classifier in $[A]$ is updated with respect to its current scaled relative accuracy κ', which is derived from the current reward prediction error ε as follows:

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha \left(\frac{\varepsilon}{\varepsilon_0}\right)^{-\nu} & \text{otherwise} \end{cases}, \tag{4.5}$$

$$\kappa' = \frac{\kappa \cdot num}{\sum\limits_{cl \in [A]} cl.\kappa \cdot cl.num}. \tag{4.6}$$

Essentially, κ measures the current absolute accuracy of a classifier using a power function with exponent ν to further prefer low error classifiers. Threshold ε_0 denotes a threshold of maximal error tolerance. That is, classifiers whose error estimate ε drops below threshold ε_0 are considered accurate. The derivation of accuracy κ with respect to ε is illustrated in Figure 4.1. The relative accuracy κ' then reflects the relative accuracy with respect to the other classifiers in the current action set. In effect, each classifier in $[A]$ competes for a limited fitness resource that is distributed dependent on $\kappa \cdot num$.

Finally, fitness estimate F is updated with respect to the current action set relative accuracy κ' as follows:

$$F \leftarrow F + \beta(\kappa' - F). \tag{4.7}$$

In effect, fitness loosely reflects the moving average, set-relative accuracy of a classifier. As before, β controls the sensitivity of the fitness.

Additionally, the action set size estimate as is updated similar to the reward prediction R but with respect to the current action set size $|[A]|$:

$$as \leftarrow as + \beta(|[A]| - as), \tag{4.8}$$

revealing a similar sensitivity to action set size changes $|[A]|$ dependent on learning rate β.

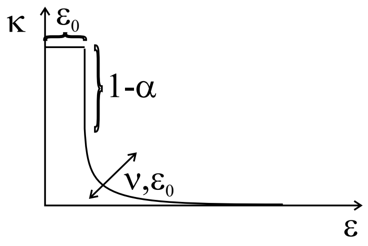

Fig. 4.1. The derivation of accuracy κ from the current reward prediction error ε has an error tolerance of ε_0. Exponent ν controls the degree of the drop off and ε_0 scales the drop off. Parameter α differentiates accurate and inaccurate classifiers further.

Parameters R, ε, and as are updated using the *moyenne adaptative modifiée* (which could be translated as adaptive average-based modification) technique (Venturini, 1994). This technique sets parameter values directly to the average of the so far encountered cases as long as the experience of a classifier is less than $1/\beta$.

Each time the parameters of a classifier are updated, experience counter exp is increased by one. Additionally, if genetic reproduction is applied to classifiers of the current action set, all time stamps ts are set to the current iteration time t.

Using the Widrow-Hoff delta rule, the reward prediction of a classifier approximates the mean reward it encounters in a problem. Thus, the reward prediction of a classifier can be approximated by the following estimate:

$$cl.R \approx \frac{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)\rho(s, cl.A)}{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)}, \tag{4.9}$$

where $p(s)$ denotes the probability that state s is presented in the problem and $p(a|s)$ specifies the conditional probability that action (or classification) a is chosen given state s.

Given that reward prediction is well-estimated, we can derive the prediction error estimate in a similar fashion:

$$cl.\varepsilon \approx \frac{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)(|cl.R - \rho(s, cl.A)|)}{\sum_{\{s|cl.C \text{ matches } s\}} p(s)p(cl.A|s)}. \tag{4.10}$$

Thus, the reward prediction estimates the mean reward encountered in a problem and the reward prediction error estimates the *mean absolute deviation* (MAD) from the reward prediction.

In a two-class classification problem, which provides 1000 reward if the chosen class was correct and 0 otherwise, we may denote the probability that a particular classifier predicts the correct outcome by p_c. Consequently, the reward prediction $cl.R$ of classifier cl will approximate $1000p_c$. Neglecting the

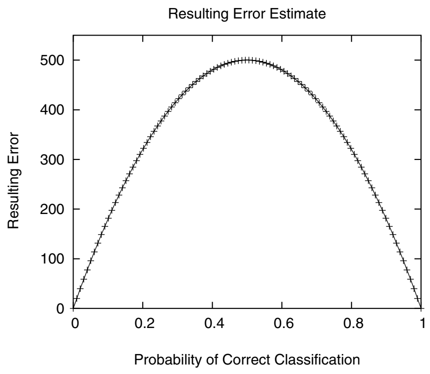

Fig. 4.2. Assuming a 1000/0 reward scheme, it can be seen that the prediction error estimate peaks at a probability of a correct classification of 0.5. Given that this is the probability of a completely general classifier, the more probable a classification is correct or wrong, the lower the error.

oscillation and consequently setting $cl.R$ equal to $1000p_c$, the following error derivation is possible:

$$cl.\varepsilon = (1000 - cl.R)p_c + cl.R(1 - p_c) = \\ = 2000(p_c - p_c^2). \tag{4.11}$$

The formula sums the two cases of executing a correct or wrong action with the respective probabilities. The result is a parabolic function for the error ε that reaches its maximum of 0.5 when $p_c(cl)$ equals 0.5 and is 0 for $p_c(cl) = 0$ and $p_c(cl) = 1$. The function is depicted in Figure 4.2. It should be noted that this parabolic function is very similar to the concept of entropy so that the prediction error estimate can also be regarded as an entropy estimate. The main idea behind this function is that given a 50/50 probability of classifying correctly, the mean absolute error will be on its highest value. The more consistently a classifier classifies problem instances correctly/incorrectly the lower the reward prediction error.

4.1.4 Rule Evolution

Besides the aforementioned covering mechanism that ensures that all actions in a particular problem instance are represented by at least one classifier, XCS applies a GA for rule evolution. Genetic reproduction is invoked in the current action set $[A]$ if the average time since the last GA application (stored in parameter ts) upon the classifiers in $[A]$ exceeds threshold θ_{GA}.

The GA selects two parental classifiers using proportionate selection where the probability of selecting classifier cl ($p_s(cl)$) is determined by its relative fitness in $[A]$, i.e. $p_s(cl) = F(cl)/\sum_{c\in[A]} F(c)$). Two offspring are generated reproducing the parents and applying crossover and mutation. Parents stay in the population competing with their offspring. Usually, we apply *free mutation*, in which each attribute of the offspring condition is mutated to the other two possibilities with equal probability. Another option is to apply *niche mutation*, which assures that the mutated classifier still matches the current problem instance.

Offspring parameters are initialized by setting prediction R to the currently received reward and reward prediction error ε to the average error in the action set. Fitness F and action set size estimate as are set to the parental values. Additionally, fitness F is decreased to 10% of the parental fitness being pessimistic about the offspring's fitness. Experience counter exp and numerosity num are set to one.

In the insertion process, *subsumption deletion* may be applied (Wilson, 1998) to stress generalization. Due to the possible strong effects of *action set subsumption* (Wilson, 1998; Butz & Wilson, 2002), we apply *GA subsumption* only. *GA subsumption* checks offspring classifiers to see whether their conditions are logically subsumed by the condition of another *accurate* ($\varepsilon < \varepsilon_0$) and *sufficiently experienced* ($exp > \theta_{sub}$) classifier in $[A]$. If an offspring is subsumed, it is not inserted in the population, but the subsumer's numerosity is increased.

The population of classifiers $[P]$ is of maximum size N. Excess classifiers are deleted from $[P]$ with probability proportional to the action set size estimate as that the classifiers occur in. If the classifier is sufficiently experienced $exp > \theta_{del}$ and its fitness F is significantly lower than the average fitness $\overline{F}$ of classifiers in $[P]$ ($F < \delta\overline{F}$), its deletion probability proportion is increased by factor $\overline{F}/F$.

4.1.5 XCS Learning Intuition

The overall learning process is illustrated in Figure 4.3. As can be seen, in contrast to the simple LCS, XCS reproduces classifiers selecting from the current action set instead of from the whole population. Several additional classifier parameters, and most explicitly the accuracy-based fitness measure F, monitor the performance of the classifier.

XCS strives to predict all possible reward values equally accurately. The amount of reward itself does not bias XCS's learning mechanism. Additionally, XCS tends to evolve a *general* problem solution since reproduction favors classifiers that are frequently active (part of an action set) whereas deletion deletes from the whole population. Among classifiers that are *accurate* (that is, the error is below ε_0) and *semantically* equally general (the classifiers are active equally often), subsumption deletion causes additional generalization pressure in that a *syntactically* more general classifier absorbs more specialized offspring classifiers.

Due to the niche reproduction in conjunction with action set size based deletion, the evolving representation is designed to stay complete. Since the

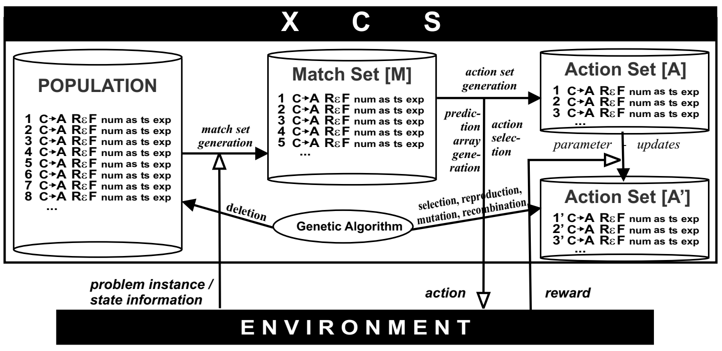

Fig. 4.3. Learning in XCS differs from simple LCSs in that the GA selects classifiers for reproduction in the current action set but for deletion in the whole population.

action set size estimate decreases in classifiers that currently inhabit underrepresented niches, their probability of deletion decreases, resulting in effective niching. Reproduction is dependent on the frequency of occurrence. In effect, XCS is designed to evolve accurate payoff predictions for all problem spaces that occur sufficiently frequently.

Together, the learning processes in XCS are designed to achieve one common goal: to evolve a *complete, maximally accurate, and maximally general* representation of the underlying payoff-map, or Q-value function. This representation was previously termed the optimal solution representation $[O]$ (Kovacs, 1996; Kovacs, 1997).

Before we analyze the XCS system in detail in the subsequent chapters, the next section provides further insights into XCS, considering learning and problem representation of XCS in a small classification problem and a small RL problem.

4.2 Simple XCS Applications

In order to better understand XCS's mechanisms this section applies XCS to several small exemplar problems. In addition to the understanding of XCS functioning, this section provides a comparison to RL and the Q-learning approximation method in XCS. First, however, we investigate XCS's performance in a simple classification problem.

4.2.1 Simple Classification Problem

In order to disregard the additional complication of reward back-propagation, characterized by the Q term derived in Equation 4.2, a big part of the analyses in the subsequent chapters focuses on classification or one-step problems. Since in such problems successive problem instances usually depend neither on each other nor on the chosen classification, each learning iteration can be treated independently.

As our simple example, we use the 6-multiplexer problem (see Appendix C). The multiplexer problem is interesting because the solution can be represented by completely non-overlapping niches and the path to the solution is of interest as well. The optimal XCS population in the 6-multiplexer is shown in Table 4.1.

Table 4.1. The optimal population $[O]$ in the 6-multiplexer problem.

Nr.	C	A	R	ε	F	Nr.	C	A	R	ε	F
1	000###	0	1000	0	1	2	000###	1	0	0	1
3	001###	0	0	0	1	4	001###	1	1000	0	1
5	01#0##	0	1000	0	1	6	01#0##	1	0	0	1
7	01#1##	0	0	0	1	8	01#1##	1	1000	0	1
9	10##0#	0	1000	0	1	10	10##0#	1	0	0	1
11	10##1#	0	0	0	1	12	10##1#	1	1000	0	1
13	11###0	0	1000	0	1	14	11###0	1	0	0	1
15	11###1	0	0	0	1	16	11###1	1	1000	0	1

It is important to notice that XCS represents both the correct classification and the incorrect classification for each problem subsolution. As mentioned before, XCS is designed to evolve a complete and accurate payoff map of the underlying problem. Thus, XCS represents each subsolution in the 6-multiplexer twice: (1) by specifying the correct condition and action combination; (2) by specifying the incorrect class as an action and predicting accurately ($\varepsilon = 0$) that the specified action is incorrect (resulting always in zero reward $R = 0$).

How might XCS evolve such an optimal population? In general, two ways may lead to success, starting either from the over-specific or the overgeneral side. Starting over-specific, the don't care probability $P_{\#}$ is set low so that initial classifiers nearly specify all l attributes in the problem. Most classifiers then are maximally accurate so that inaccurate classifiers quickly disappear since selection favors accurate classifiers. Mutation mainly results in generalized offspring since mainly specified attributes will be chosen for mutation. If mutation results in an inaccurate, overgeneralized classifier, the accuracy and thus fitness of the offspring is expected to drop and the classifier will have hardly any reproductive events and consequently will soon be deleted. Since more general classifiers will be more often part of an action set, more general, accurate classifiers undergo more reproductive events and thus propagate faster. Thus, the process is expected to evolve the accurate, maximally general solution as the final outcome.

Despite this appealing description, we will see that a start from the overspecialized side is usually undesirable because of the large requirements on population size. In essence, when starting completely specialized, the popula-

Table 4.2. The path to an optimal solution from the overgeneral side in the 6-multiplexer problem.

Nr.	C	A	R	ε	Nr.	C	A	R	ε
1	`######`	0	500.0	500.0	2	`######`	1	500.0	500.0
3	`1#####`	0	500.0	500.0	4	`1#####`	1	500.0	500.0
5	`0#####`	0	500.0	500.0	6	`0#####`	1	500.0	500.0
7	`##1###`	0	375.0	468.8	8	`##1###`	1	625.0	468.8
9	`##0###`	0	625.0	468.8	10	`##0###`	1	375.0	468.8
11	`##11##`	0	250.0	375.0	12	`##11##`	1	750.0	375.0
13	`##00##`	0	250.0	375.0	14	`##00##`	1	750.0	375.0
15	`0#1###`	0	250.0	375.0	16	`0#1###`	1	750.0	375.0
17	`0#0###`	0	750.0	375.0	18	`0#0###`	1	250.0	375.0
19	`0#11##`	0	0.0	0.0	20	`0#11##`	1	1000.0	0.0
21	`001###`	0	0.0	0.0	22	`001###`	1	1000.0	0.0
23	`10##1#`	0	0.0	0.0	24	`10##1#`	1	1000.0	0.0
25	`000###`	0	1000.0	0.0	26	`000###`	1	0.0	0.0
27	`01#0##`	0	1000.0	0.0	28	`01#0##`	1	0.0	0.0

tion size needs to be chosen larger than 2^l and thus exponentially in problem length, which is obviously a highly undesirable requirement.

Thus, XCS usually starts its search for an optimal solution representation from the overgeneral side. In the most extreme case, the don't care probability may be set to $P_{\#} = 1$ and XCS starts its search with the two completely general classifiers `######`$\rightarrow$`0` and `######`$\rightarrow$`1`. Mutation is then initially responsible for the introduction of more specialized classifiers. However, mutation alone is usually not sufficient since more general classifiers are part of more action sets undergoing more reproductive events, on average resulting in an overall generalization pressure. Thus, classifiers need to be propagated that are more accurate. Table 4.2 shows the resulting expected average R and ε values for progressively more specialized classifiers. Fitness is not shown since fitness depends on the classifier distribution in the current population.

Note how initially the specialization of the value bits results in a smaller error and thus a larger accuracy. Once a value bit is specified, the specialization of an address bit or an additional value bit has an equal effect on error. Thus, the evolutionary process is initially guided towards specializing value bits. Soon, however, the beneficial specification of an address bit takes over and completely accurate classifiers evolve.

Note how XCS explores both sides of the solution spectrum: the more incorrect as well as the more correct classification side. In effect, a complete payoff map is evolved dependent on if better classifiers (lower error classifiers) have enough time to propagate and a complete solution can be supported. Subsequent chapters investigate these issues in further detail.

4.2.2 Performance in Maze 1

Compared to a classification problem, an RL problem poses the additional complication of back-propagating reward appropriately. Due to the generalized, distributed representation of the Q-function in XCS, additional complications might arise caused by inappropriate reward propagations from inaccurate, young, or overgeneralized classifiers. Chapter 10 investigates performance in RL problems in much more detail.

In this section, we investigate XCS learning and solution representation in the Maze 1 problem shown in Figure 2.3 on page 17. The question is, if XCS can be expected to solve this problem and evolve a complete, but generalized representation of the underlying Q-table.

Lanzi (2002) provides a detailed comparison of XCS with Q-learning from the RL perspective. The investigations show that if no generalization is allowed, XCS essentially mimics Q-learning. Each classifier corresponds to exactly one entry in the Q-table (shown in Table 2.2 on page 20 for the Maze 1 case) and the Q-learning update function:

$$Q(s,a) \leftarrow Q(s,a) + \beta(r + \gamma \mathtt{max}_{a'} Q(s',a') - Q(s,a)) \tag{4.12}$$

is equivalent to the XCS update function for the reward prediction

$$R \leftarrow R + \beta(r + \gamma \mathtt{max}_{A \in \mathcal{A}} P^{t+1}(A) - R) \tag{4.13}$$

in that the reward prediction values R coincide with the Q-values $Q(s,a)$ since the condition of a classifier specifies exactly one state s and one action a. Thus, the prediction array coincides with Q-value entries, since for each prediction array entry exactly one classifier applies, so that the entry is equivalent to the R value of the classifier (see Equation 4.1), which is equivalent to the Q-value as outlined above. Thus, without generalization, XCS is a tabular Q-learner where each tabular entry is represented by a distinct rule.

What is expected to happen in the case when generalization is allowed? What does the perfect representation look like? The optimal population in the Maze 1 problem is shown in Table 4.3. In comparison to the Q-table in Table 2.2, we see that XCS is able to represent the maze in a much more compact form requiring only 19 classifiers to represent a Q-table of size 40. Note how XCS is generalizing over the state space with respect to each action as indicated by the action order in Table 4.3. Essentially, since each state combination is representable with a different action code (for example, the condition of classifier one specifies that it matches in situations A or E), XCS generalizes over states that yield identical Q-values with respect to a specific action. Even in our relatively simple environment in which there are only three different Q-values possible (810, 900, and 1000) generalization is effective. Given a larger state space, further generalizations over different states are expectable. In sum, XCS learns a generalized Q-table representation specializing those attributes that are relevant for the distinction of different Q-values.

Table 4.3. The optimal population $[O]$ in the Maze 1 problem.

Nr.	C	A	R	ε	Nr.	C	A	R	ε
1	`11#####1`	$\uparrow$	810	0	2	`1#0###0#`	$\uparrow$	900	0
3	`0#######`	$\uparrow$	1000	0					
4	`11#####1`	$\nearrow$	810	0	5	`#10###0#`	$\nearrow$	900	0
6	`#0######`	$\nearrow$	1000	0					
7	`##0####1`	$\rightarrow$	900	0	8	`11####0#`	$\rightarrow$	810	0
9	`11#####1`	$\searrow$	810	0	10	`##0###0#`	$\searrow$	900	0
11	`11#####1`	$\downarrow$	810	0	12	`##0###0#`	$\downarrow$	900	0
13	`11#####1`	$\swarrow$	810	0	14	`##0###0#`	$\swarrow$	900	0
15	`1#0####1`	$\leftarrow$	810	0	16	`#1####0#`	$\leftarrow$	900	0
17	`11#####1`	$\nwarrow$	810	0	18	`##0###01`	$\nwarrow$	900	0
19	`#######0`	$\nwarrow$	1000	0					

The question remains how XCS learns the desired complete, accurate, and maximally problem solution shown in Table 4.3. Dependent on the initialization procedure, classifiers might initially randomly distinguish some states from others. Additionally, the back-propagated reward signal is expected to fluctuate significantly. As in Q-learning, XCS is expected to progressively learn starting from those state-action combinations that yield actual reinforcement. Since these transitions result in an exact reward of 1000, classifiers that identify these cases quickly become accurate and thus will be reproduced most of the time they are activated. Once, the 1000 reward cases are stably represented, back-propagation becomes more reliable and the next reward level (900) can be learned accurately and so forth. Thus, as in Q-learning, reward will be spread backward starting from the cases that yield actual reward. In chapter 10, we show how XCS's performance can be further improved to ensure a more reliable and stable solution in RL problems applying gradient-based update techniques. In sum, due to the accuracy-based fitness approach, XCS is expected to evolve a representation that covers the whole problem space of Maze 1, yielding classifiers that comprise the maximal number of states in their conditions to predict the Q-value accurately with respect to the specified action.

We still have not answered how XCS evolves a *maximally general* problem representation. For example, Classifier 12 specifies that if moving south and if currently located in state B, C, or D, then a reward of 900 is expected. Classifier 12 is *maximally general* in that any further generalization of the classifier's condition will result in an inaccurate prediction (that is, state A or E will be accepted by the condition part in which a south action yields a reward of 810). However, Classifier 12 can be further specialized still matching in all three states B, C, and D. In essence, in all three states the positions to the southeast, south, and southwest are blocked by obstacles so that the specification of obstacles in these positions is redundant. Only subsumption is able to favor the syntactically more general Classifier 12. Thus, due to the sparse problem instance coverage (only five perceptions possible) of the whole problem space (the coding allows $2^l = 2^8 = 256$ different problem instances), only few (semantic) generalizations over different states are possible. Syntactic generalizations, which identify redundant specializations, are much more important. Thus, subsumption is a major factor for evolving maximally general classifiers in environments with sparse problem input space $\mathcal{S}$ coverage. If Maze 1 was noisy and syntactic generalization was desired, the error threshold ε_0 may need to be increased to enable subsumption, as suggested in Lanzi (1999c).

4.3 Summary and Conclusions

This chapter introduced the accuracy-based learning classifier system XCS. As all (Michigan-style) LCSs, XCS is an online learner applying RL techniques for rule evaluation and action decisions and GA techniques for rule discovery. In contrast to traditional LCSs, XCS derives classifier fitness from the *accuracy of reward prediction* instead of from the reward prediction value directly. The reward updates can be compared to Q-learning updates. Thus, XCS is designed to learn a generalized representation of the underlying Q-function in the problem.

XCS's *niche reproduction* in combination with population-wide deletion results in an *implicit, semantic generalization pressure* that favors classifiers that are more frequently active. Additionally, subsumption deletion favors accurate classifiers that are *syntactically* more general. Niche reproduction in combination with population-wide deletion based on the action set size estimates results in effective problem niching striving to evolve and maintain a complete problem solution.

In effect, XCS is designed to evolve a *complete, maximally accurate, and maximally general problem solution* represented by a population of classifiers. Each classifier defines a problem subspace in which it predicts the corresponding Q-value accurately.

The application to two small problems showed which solution representation XCS is designed to evolve and how it might be evolved. Starting from the overgeneral side, higher accurate classifiers need to be detected and propagated. Starting from the over-specific side, generalizations due to mutations boil the representation down to the desired accurate, maximally general one. However, when starting over-specialized, the required population size needs to grow exponentially in problem length l given that all problem instances are equally likely to occur.

In problems in which only a few samples are available so that the problem space is covered only sparsely, syntactic generalization and thus subsumption becomes more important since the usual generalization pressure due to more frequent reproductive opportunities may not apply. However, subsumption only works for accurate classifiers so that a proper choice of the error threshold ε_0, and thus an appropriate noise tolerance, becomes more important.

While the base XCS system introduced in this section learns a generalized representation of the underlying Q-value function, it should be noted that XCS is not limited to approximating Q-functions. In fact, XCS can be modified yielding a general, online learning *function approximation technique* (Wilson, 2001a). Due to the rule-based structure, XCS is designed to partition its space dependent on the representation of classifier conditions. Each classifier then approximates, or predicts, the actual function value in its defined subspace. In the base XCS, conditions define hyperrectangles as subspaces and predict constant reward values—consequently applying piece-wise constant function approximation. Wilson (2001a) experimented with piecewise linear approximations. Other prediction methods are imaginable as well, such as the most recently introduced polynomial predictions (Lanzi, Loiacono, Wilson, & Goldberg, 2005). Similarly, XCS can be endowed with other condition structures such as S-expressions (Lanzi, 1999b). As we will see, dependent on the problem structure at hand, the most suitable condition structure can be expected to be most advantageous for learning with XCS.