Rule Fitness and Pathology in Learning Classifier Systems

Tim Kovacs kovacs@cs.bris.ac.uk
Department of Computer Science, University of Bristol, Bristol, BS8 1UB, UK

Abstract

It has long been known that in some relatively simple reinforcement learning tasks traditional strength-based classifier systems will adapt poorly and show poor generalisation. In contrast, the more recent accuracy-based XCS, appears both to adapt and generalise well. In this work, we attribute the difference to what we call strong over general and fit over general rules. We begin by developing a taxonomy of rule types and considering the conditions under which they may occur. In order to do so an extreme simplification of the classifier system is made, which forces us toward qualitative rather than quantitative analysis. We begin with the basics, considering definitions for correct and incorrect actions, and then correct, incorrect, and overgeneral rules for both strength and accuracy-based fitness. The concept of strong overgeneral rules, which we claim are the Achilles' heel of strength-based classifier systems, are then analysed. It is shown that strong overgenerals depend on what we call biases in the reward function (or, in sequential tasks, the value function). We distinguish between strong and fit overgeneral rules, and show that although strong overgenerals are fit in a strength-based system called SB–XCS, they are not in XCS. Next we show how to design fit overgeneral rules for XCS (but not SB–XCS), by introducing biases in the variance of the reward function, and thus that each system has its own weakness. Finally, we give some consideration to the prevalence of reward and variance function bias, and note that non-trivial sequential tasks have highly biased value functions.

Keywords

Learning Classifier Systems, Reinforcement Learning, Credit Assignment, XCS.

1 Introduction

When applied to reinforcement learning, Learning Classifier Systems (LCS) evolve sets of rules in order to maximise the return they receive from their task environment. They employ a genetic algorithm to generate rules, and to do so must evaluate the fitness of existing rules. In order for the Genetic Algorithm (GA) to produce rules which are better adapted to the task, rule fitness needs somehow to be connected to the rewards received by the system – a credit assignment problem. Precisely how to relate LCS performance to rule fitness has been the subject of much research, and is of great significance because adaptation of rules and LCS alike depends on it.

This work undertakes an analysis of the causes and effects of certain rule pathologies in traditional strength-based LCS (§2.3) and traces them ultimately to the relation between LCS performance and rule fitness – i.e., to the credit assignment system. We examine situations in which less desirable rules can achieve higher fitness than more desirable rules, which constitutes a mismatch between the goal of the LCS as a whole (adaptation to a task) and the goal of the GA (evolution of high-fitness rules).

To study rule pathology we undertake an analysis of what types of rules, and relationships between rules, are possible. Developing earlier work by Cliff and Ross (1995)

and Lettau and Uhlig (1994, 1999), the notion of *strong overgeneral rules* (strong overgenerals, for short) is studied and it is shown exactly what requirements must be met for them to arise in both strength and accuracy-based LCS. In order to compare the two approaches we use accuracy-based XCS and its strength-based twin SB–XCS, which were designed to differ as little as possible, allowing us to isolate the effects of the fitness calculation on performance. The analysis is undertaken using a number of simplifying assumptions outlined in §4, and deals first with non-sequential tasks.

This paper argues that different definitions of overgenerality and strong overgenerality are appropriate for the two types of LCS (§5.3). Minimal conditions and tasks which will support strong overgeneral rules are presented (sections §6, §7 and §8), their dependence on the reward function is demonstrated (§6.1), and certain theorems regarding their prevalence are proved under simplifying assumptions (§7.2 and §8). It is shown that XCS and SB–XCS have different kinds of tolerance for biases (see §6.1) in reward functions, and (within the context of various simplifying assumptions) to what extent we can bias them without producing strong overgenerals (§8.2). It is also shown what kinds of tasks will not produce strong overgenerals even without our simplifying assumptions (§8.1 and §12.6).

Next fit overgeneral rules are distinguished (§5.5) and it is shown how XCS and SB–XCS differ in their response to them (§9). Although rules which are fit overgenerals for SB–XCS are not necessarily fit for XCS, we show that XCS can suffer from fit overgenerals (§10). Indeed, we show how to produce rules which are fit overgenerals for XCS but not SB–XCS, and in doing so demonstrate that each has a class of problems to which it fails to adapt. Following this, the concept of a strong *under*general rule is introduced and it is noted that a generalisation bias in fitness is needed to avoid them (§11).

In §12 we proceed to consider the more complex case of sequential tasks, and show that sequential tasks amplify difficulties with strong overgenerals. We conclude (§13) that SB–XCS is unsuitable for non-trivial sequential tasks and that it appears to have no niche (i.e., no useful domain of application), unless fitness sharing can resolve its problems (§14.1). This work concludes with consideration of the value of the approach taken and possible extensions (§14) and some final comments (§15).

2 Background

The arguments presented here require some basic knowledge of reinforcement learning, including the notions of reward functions, value functions, the Q-update, and discounting return. Each of these is introduced as needed, and the reader is referred to Sutton and Barto (1998) for a more complete introduction. We make use of XCS and SB–XCS, and the details of their fitness calculations are relevant to the results presented later. We introduce them in the following sections, and more details are available in (Kovacs, 2002).

2.1 Classifier Systems for Reinforcement Learning

Reinforcement learning consists of cycles in which a learning agent is presented with an input describing the current environmental state, responds with an action and receives some reward as an indication of the value of its action. The reward received is defined by the *reward function* R, which maps state-action pairs to the real number line, and which is part of the problem definition (Sutton and Barto, 1998). For simplicity we initially consider only *non-sequential* tasks, in which the agent's actions do not affect which states it visits in the future. The goal of the agent is to maximise the rewards it

receives, and, in non-sequential tasks, it can do so in each state independently. In other words, it need not consider sequences of actions in order to maximise reward.

When an LCS receives an input it forms the *match set* [M] of rules whose conditions match the environmental input. (The two systems we consider here are *stimulus-response* LCS, that is, they lack an internal message list.) The LCS then selects an action from among those advocated by the rules in [M]. The subset of [M] which advocates the selected action is called the *action set* [A]. Occasionally the LCS will trigger a reproductive event, in which it calls upon the GA to modify the population of rules.

We will consider LCS in which, on each cycle, only the rules in [A] are updated based on the reward received – rules not in [A] are not updated.

2.2 The Standard Ternary LCS Language

A number of representations have been used with LCS, in particular a number of variations based on binary and ternary strings. Using what we'll call the *standard ternary LCS language* each rule has a single condition and a single action. Conditions are fixed length strings from $\{0,1,\#\}^l$, while rule actions and environmental inputs are fixed length strings from $\{0,1\}^l$. In all problems considered here $l = 1$.

A rule's condition c matches an environmental input m if for each character m_i the character in the corresponding position c_i is identical or the wildcard (#). The wildcard is the means by which rules generalise over environmental states; the more #s a rule contains the more general it is. Since actions do not contain wildcards the system cannot generalise over them.

2.3 Strength-based and Accuracy-based Fitness

Although the fitness of a rule is determined by the rewards the LCS receives when it is used, LCS differ in how they calculate rule fitness. In traditional strength-based systems (see, e.g., Goldberg1989; Wilson1994), the fitness of a rule is called its *strength*. This value is used in both action selection and reproduction. In contrast, the more recent accuracy-based XCS (Wilson, 1995) maintains separate estimates of rule utility for action selection and reproduction.

One of the goals of this work is to compare the way strength and accuracy-based systems handle overgeneral and strong overgeneral rules. To do so, we'll compare accuracy-based XCS with a strength-based LCS called SB–XCS which differs as little as possible from XCS, and which closely resembles Wilson's ZCS (Wilson, 1994). Specifically, SB–XCS updates rule strengths as follows:[1]

Strength (also called prediction):

$$p_j \leftarrow p_j + \beta(P - p_j) \tag{1}$$

where p_j is the prediction (or strength) of rule j, and $0 < \beta \leq 1$ is a constant controlling the learning rate. In non-sequential tasks, P is the immediate reward from the environment. In sequential tasks, P is analogous to the discounted maximum Q-value of the successor state in Q-learning (see equation 6). SB–XCS uses the same strength value for both action selection and reproduction. That is, the fitness of a rule in the GA is simply its strength.

XCS uses this same update to calculate rule strength, and uses strength in action selection, but goes on to derive other statistics from it. In particular, from strength it

[1] Wilson (1994) refers to strength as *prediction* because he treats it as a prediction of the reward that the system will receive when the rule is used. We will use the terms interchangeably.

derives the accuracy of a rule, which it uses as the basis of its fitness in the GA. This is achieved by updating a number of parameters as follows (see Wilson, 1995, for more). Following the update of a rule's strength p_j, we update its prediction error ε_j.

Prediction error:

$$\varepsilon_j \leftarrow \varepsilon_j + \beta\left(|\overbrace{P - p_j}^{Error}| - \varepsilon_j\right) \tag{2}$$

Next we calculate the rule's accuracy κ_j:

Accuracy:

$$\kappa_j = \begin{cases} 1 & \text{if } \varepsilon_j < \varepsilon_o \\ \alpha(\varepsilon_j/\varepsilon_o)^{-v} & \text{otherwise} \end{cases} \tag{3}$$

where $0 < \varepsilon_o$ is a constant controlling the tolerance for prediction error and $0 < \alpha < 1$ and $0 < v$ are constants controlling the rate of decline in accuracy when ε_o is exceeded. Once the accuracy of all rules in [A] has been updated we update each rule's relative accuracy κ'_j :

Relative Accuracy:

$$\kappa'_j = \frac{\kappa_j \cdot numerosity(j)}{\sum_{x \in [A]} \kappa_x \cdot numerosity(x)} \tag{4}$$

where $numerosity(j)$ is the number of copies of a rule represented by a single macro-classifier j (see Wilson, 1995). Finally, each rule's fitness F_j is updated:

Fitness:

$$F_j \leftarrow F_j + \beta(\kappa'_j - F_j) \tag{5}$$

To summarise, the XCS updates treat the strength of a rule as a prediction of the reward to be received, and maintain an estimate of the error ε_j in each rule's prediction. An accuracy score κ_j is calculated based on the error as follows. If error is below some threshold ε_o the rule is fully accurate (has an accuracy of 1), otherwise its accuracy drops off quickly. The accuracy values in the action set [A] are then converted to relative accuracies (the κ'_j update), and finally each rule's fitness F_j is updated toward its relative accuracy. To simplify, in XCS fitness is an inverse function of the error in reward prediction, with errors below ε_o being ignored entirely.

Sequential Tasks For sequential tasks, rules in the previous time step's action set $[\mathrm{A}]_{-1}$ are updated toward the sum of the previous time step's reward and the discounted maximum of the current time step's strengths / predictions:

$$P = r_{t-1} + \gamma \max_i P(a_i) \tag{6}$$

where r_{t-1} is the immediate reward on the previous time step, $0 \leq \gamma \leq 1$ is the *discount rate* which weights the contribution of the next time step to the value of P, and $P(a_i)$ is the system prediction for action a_i (defined in §2.4).

Summary of Strength and Accuracy In short, in XCS accuracy evaluates the utility of generalisation, while strength evaluates the utility of acting. In SB–XCS, in contrast, strength plays both roles.

2.4 Action Selection

In XCS, a rule's contribution to system prediction[2] is its prediction weighted by its fitness (so that less fit rules have less weight):

$$P(a_i) = \frac{\sum_{c \in [\mathrm{M}]_{a_i}} F_c \cdot p_c}{\sum_{c \in [\mathrm{M}]_{a_i}} F_c} \tag{7}$$

where $[\mathrm{M}]_{a_i}$ is the subset of the match set [M] advocating action a_i, F_c is the fitness of rule c and p_c is its prediction.

In SB–XCS, however, a rule's fitness is its strength, so there is no separate fitness parameter to factor into the calculation – and low prediction rules already have less weight. However, we do need to weight SB–XCS's prediction by numerosity, so that macroclassifiers have influence equal to the equivalent number of microclassifiers. The XCS update does not include numerosity because XCS's fitness already includes numerosity, thanks to the relative accuracy update (4). Removing fitness from equation (7) and factoring in numerosity we obtain the *System Strength*:

$$S(a_i) = \sum_{c \in [\mathrm{M}]_{a_i}} p_c \cdot numerosity(c) \tag{8}$$

In preparation for action selection, SB–XCS constructs a system strength array using (8), just as XCS constructs a system prediction array using (7). Note, however, that the two differ in that the system strength (8) for an action is *not* a prediction of the reward to be received for taking it. For example, suppose that in a given state action 1 receives a reward of 1000, and that the only matching macroclassifier advocating action 1 has strength 1000 and numerosity 2. The system strength for action 1 is $P(a_1) = 1000 \cdot 2 = 2000$, twice the actual reward since there are two copies of the rule.

In order to estimate the return for an action, we must divide the system prediction by the numerosity of the rules which advocate it. For this purpose, we define the *System Prediction* in SB–XCS as:

$$\begin{aligned} P(a_i) &= \frac{S(a_i)}{\sum_{c \in [\mathrm{M}]_{a_i}} numerosity(c)} \\ &= \frac{\sum_{c \in [\mathrm{M}]_{a_i}} p_c \cdot numerosity(c)}{\sum_{c \in [\mathrm{M}]_{a_i}} numerosity(c)} \end{aligned} \tag{9}$$

[2]The estimate of the return for taking a given action – essentially a Q-value in reinforcement learning terms.

The system prediction is needed to calculate the target for the Q-update in sequential tasks (6).

In summary, whereas XCS uses system prediction for both action selection and the Q-update, SB–XCS uses system strength for the former and system prediction only for the latter.

2.5 XCS, SB–XCS and other LCS

SB–XCS is not simply a straw man for XCS to outperform. It is a functional LCS, and is capable of solving some problems well. (For example, its performance on the 6 multiplexer task is similar to XCS's – see Kovacs, 2002.) SB–XCS's value is that we can study when and why it fails, and we can attribute any difference between its performance and that of XCS to the difference in fitness calculation. See (Kovacs, 2002) for full details of both XCS and SB–XCS.

3 Known Problems with Strength LCS

In this section we review a number of known problems with strength LCS, which will serve as the starting point for the analysis presented later in this work.

3.1 Overgeneral Rules

Dealing with *overgeneral rules* – rules which are simply too general – is a fundamental problem for LCS. Such rules may specify the desired action in a subset of the states they match, but, by definition, not in all states, so relying on them harms performance. E.g., an overgeneral which matches 10 states may be correct in as many as 9 or as few as 1. Even overgenerals which are most often correct are (by definition) sometimes incorrect, so using them can harm the performance of the system.

3.2 Greedy Classifier Creation

Another problem faced by some LCS is what Cliff and Ross referred to as *greedy classifier creation* (Cliff and Ross, 1995; Wilson, 1994). To obtain better rules, a classifier system's GA allocates reproductive events preferentially to rules with higher fitness. This is simply the application of selective pressure in reproduction, one of the components of the evolutionary process, and of itself is not a problem. However, in many classifier systems a rule's fitness depends on the magnitude of the reward it receives. In such systems rules which match in higher-rewarding parts of the task will reproduce more than others. If the bias in reproduction of rules is strong enough there may be too few rules, or even no rules, matching low-rewarding states. (In the latter case, we say there's a gap in the rules' covering map of the input / action space.)

3.3 Strong Overgeneral Rules

Cliff and Ross (1995) showed that the strength-based ZCS can have serious difficulty even with simple sequential tasks. They attributed ZCS's difficulties to the two problems above, and, in particular, to their interaction, an effect the author refers to (Kovacs, 2000; Kovacs, 2001) as the problem of *strong overgeneral rules*. The interaction occurs when an overgeneral rule acts correctly in a high reward state and incorrectly in a low reward state. The rule is overgeneral because it acts incorrectly in one of the states, but at the same time it prospers because of greedy classifier creation and the high reward it receives in the other state.

Lettau and Uhlig (1994; 1999) independently discovered strong overgeneral rules using a very different approach from Cliff and Ross's. As they put it:

> "... a suboptimal rule might dominate the optimal if it is applicable only in "good" states of the world: bad decisions in good times can "feel better" than good decisions in bad times." (Lettau and Uhlig, 1999) p. 153.

Sequential and Non-sequential Tasks Although both Cliff and Ross and Lettau and Uhlig dealt exclusively with sequential tasks, the problems discussed above clearly also apply in non-sequential tasks, as this work will demonstrate. In fact, examples of trivial non-sequential tasks which produce strong overgenerals will be shown, and it is in these cases that analysis is simplest.

Significance The proliferation of strong overgenerals can be disastrous for the performance of a classifier system: such rules are unreliable, but outweigh more reliable rules when it comes to action selection. Worse, they may prosper under the influence of the GA, and may even reproduce more than reliable but low-rewarding rules, possibly driving them out of the population. For these reasons, and for their prevalence, strong overgenerals (and the related fit overgenerals of §5.5) are a major difficulty – perhaps the major difficulty – for strength-based classifier systems.

4 Methodology for Rule Type Analysis

Classifier systems are complex systems and analysis of their behaviour can be quite difficult. To make our analysis more tractable we'll make a number of simplifications, perhaps the greatest of which will be to study very small tasks. Although very small, these tasks illustrate different types of rules and the effects of different fitness definitions on them – indeed, they illustrate them better for their simplicity.

Another great simplification will be to deal initially with the much simpler case of non-sequential tasks rather than sequential ones. Sequential tasks present their own difficulties, but those present in the non-sequential case persist in the more complex sequential case; after all, non-sequential tasks are just the special case of sequential task in which $\gamma = 0$. Study of non-sequential tasks can uncover fundamental features of the systems under consideration while limiting the complexity which needs to be dealt with. Consequently, the analysis of rule types considers non-sequential tasks, and sequential tasks are dealt with only later in §12.

To further simplify matters we'll remove rule discovery from the picture and enumerate all possible classifiers for each task, which is trivial given the small tasks we'll consider. This effectively leaves us with something like a tabular Q-learner, where each entry in the table corresponds to a rule in the ternary language (see Kovacs, 2002). In other words, some table entries aggregate elementary states, unlike in a standard tabular Q-learner. This approach simplifies matters since rule discovery is no longer an issue, and the behaviour of all possible rules is considered simultaneously. We'll restrict our considerations to the standard ternary LCS language because it is the most commonly used and because we are interested in fitness calculations and the ontology of rules, not in their representation.

At present we are concerned with rule type analysis; we would like to know, for example, the conditions under which it is possible for strong overgeneral rules to occur. We will not, however, normally consider the dynamic behaviour of rules over time. For example, we will not consider how the strength of a rule changes from an initial value to a value which reflects its utility (as determined by the credit assignment system).

Instead, we'll consider the steady state values of already adapted (i.e., evaluated) rules (until §9, where we will see how fitness changes over time). In particular, we will not consider the effect of the learning rate β on rule updates: we assume it is declined appropriately so that in the limit rule strengths approach the expected values shown.

Similarly, we are not interested in fluctuations in a rule's strength due to stochastic effects, and so we will consider only the expected values of rules in our calculations, and not deviations from expectations. In fact, we will consider deterministic reward functions, although by considering expected values we could compensate for stochasticity in the reward function. For our analysis of rules types, however, deterministic reward functions suffice.

As a final simplification we'll assume that, in all tasks, states and actions occur equiprobably. That is, on any time step the LCS has the same chance of sensing any of the possible environmental states, and it chooses an action at random. This makes the calculation of steady state strengths particularly simple. For example, figure 1 defines a simple task with two states and two actions, and a reward associated with each state-action pair. Figure 2 lists all possible rules for this task, along with their expected strengths. Since all states and all actions occur with equal probabilities, a rule's expected strength is simply the average of the rewards for the state-actions it matches.

4.1 What can this Sort of Analysis Tell us?

These simplifications reduce a complex dynamic system – the interaction of an LCS with a task – to a very simple static model, in which each rule has a single fixed expected strength. Because we have simplified matters so much, there is much that such a model cannot tell us, e.g., about the dynamic behaviour of rules. In fact, removing rule discovery and choosing actions at random does not leave us with much of a classifier system and our simplifications mean that any quantitative results we obtain do not apply to any realistic applications of an LCS. However, because the model is so simple it is amenable to the analysis we will perform. In particular, this approach seems well suited to the qualitative study of rule ontology, and it will give us a qualitative sense of the behaviour of two types of LCS. §5 contains examples of this approach.

4.2 Default Hierarchies

Default hierarchies have not been included in the analysis presented here because XCS and SB–XCS do not support them. Default hierarchies are potentially significant in that they may allow strength LCS to overcome some of the difficulties with strong overgeneral rules we will show them to have. If so, this would increase both the significance of default hierarchies and the significance of the well-known difficulty of finding and maintaining them.

4.3 Fitness Sharing

Like default hierarchies, fitness sharing is a potential means for strength-based systems to escape problems with strong overgenerals. Its analysis is, unfortunately, beyond the scope of this work, and it is left as an important direction for future work. The incorporation of fitness sharing in SB–XCS would alter many, if not most, of the results in this work. It would also, however, greatly complicate the analysis presented here, and it is unlikely that the results obtained here would have been possible had fitness sharing been included in the analysis. This work is appropriately seen as a first step which identifies fundamental rule types; the effect of fitness sharing on them must await future work.

5 Analysis of Rule Types

5.1 Some Notation

Some simple notation will prove useful in our analysis of rule types.

- A Boolean target function f is a total function on a binary bit string, that is $f : \{0,1\}^n \rightarrow \{0,1\}$.
- Classifiers are constant partial functions, that is, they map some subset of the domain of f to either 0 or 1. Classifiers are constant because, using the standard ternary language, they always advocate the same action regardless of their input.

As a shorthand, and to approximate Sutton and Barto's reinforcement learning notation (Sutton and Barto, 1998), we define $\mathcal{S}$ = domain and $\mathcal{A}$ = range when dealing with classifiers and target functions. That is, a task's state is an element of $\mathcal{S}(f)$ and a classifier system's action is an element of $\mathcal{A}(f)$, where f is a target function. The states matched by a classifier c form the set $\mathcal{S}(c)$, and the action advocated by c is $\mathcal{A}(c)$.

Note that f merely defines the state-action space. The learning task an LCS faces is defined by a reward function defined over this state-action space.

5.2 Correct and Incorrect Actions

Since the goal of a reinforcement learning agent is to maximise the rewards it receives, it's useful to have terminology which distinguishes between actions which do so and those which do not:

Correct action: In any given state the agent must choose from a set of available actions. A correct action is one which results in the maximum reward possible for the given state and set of available actions.

That is, an action c is correct, correct(s,c), for a state s with respect to a reward function R iff:

$$\forall a \;\; R(s,a) \leq R(s,c)$$

Incorrect action: One which does not maximise reward.

Figure 1 defines a simple non-sequential task, in which for state 0 the correct action is 0, while in state 1 both actions 0 and 1 are correct. Note that an action is correct or incorrect only in the context of a given state and the rewards available in it.

5.3 Overgeneral Rules

Figure 2 shows all possible rules for the task in figure 1 using the standard ternary language. Each rule's expected strength is also shown, using the simplifying assumption of equiprobable states and actions from §4. The classification shown for each rule will eventually be explained in sections §5.3.2 and §5.3.3.

We're interested in distinguishing overgeneral from non-overgeneral rules. Rules A, B, C and D are clearly not overgeneral, since they each match only one input. What about E and F? So far we haven't explicitly defined overgenerality, so let's make our implicit notion of overgenerality clear:

Overgeneral rule: A rule O from which a superior rule can be derived by reducing the generality of O's condition.

State	Action	Reward
0	0	1000
1	0	500

State	Action	Reward
0	1	500
1	1	500

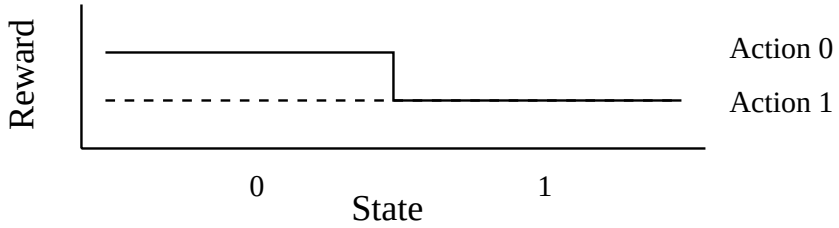


Figure 1: Reward function for a simple task.

Rule	Cond.	Action	E[Strength]	Strength Classification	Accuracy Classification
A	0	0	1000	Cons. Correct	Accurate
B	0	1	500	Cons. Incorrect	Accurate
C	1	0	500	Cons. Correct	Accurate
D	1	1	500	Cons. Correct	Accurate
E	#	0	750	Cons. Correct	Overgeneral
F	#	1	500	Overgeneral	Accurate

Figure 2: All possible classifiers for the simple task in figure 1 and their classifications using strength-based and accuracy-based fitness. ("Cons." stands for "consistently".)

This definition seems clear, but relies on our ability to evaluate the superiority of rules. That is, to know whether a rule X is overgeneral, we need to know whether there is any possible Y, some more specific version of X, which is superior to X. How should we define superiority?

5.3.1 Are Stronger Rules Superior Rules?

Can we simply use fitness itself to determine the superiority of rules? After all, this is the role of fitness in the GA. In other words, let's say X is overgeneral if some more specific version Y is fitter than X.

In SB–XCS, our strength-based system, fitter rules are those which receive higher rewards, and so have higher strength. Let's see if E and F are overgeneral using strength to define the superiority of rules.

Rule E. The condition of E can be specialised to produce A and C. C is inferior to E (it has lower strength) while A is superior (it has greater strength). Because A is superior, E is overgeneral.

This doesn't seem right – intuitively E should *not* be overgeneral, since it is correct in both states it matches. In fact all three rules (A, C and E) advocate only correct actions, and yet A is supposedly superior to the other two. This seems wrong since E subsumes A and C, which suggests that, if any of the three is more valuable, it is E.

Rule F. The condition of F can be specialised to produce B and D. Using strength as our value metric all three rules are equally valuable, since they have the same expected strength, so F is not overgeneral.

This doesn't seem right either – surely F *is* overgeneral since it is incorrect in state 0. Surely D should be superior to F since it is always correct. Clearly using strength as our value metric doesn't capture our intuitions about what the system should do. To define the value of rules let's return to the goal of the LCS, which is to maximise the reward it receives, which in turn means acting correctly in each state. It is the correctness of its actions which determines a rule's value, rather than how much reward it receives.

Recall that rule strength is derived from reward. Strength is a measure of how good – *on average* – a rule is at obtaining reward. Using strength as fitness in the GA, we will evolve rules which are – *on average* – good at obtaining reward. However, many of these rules will actually perform poorly in some states, and only achieve good average performance by doing particularly well in other states. These rules are overgeneral.

To maximise rewards, we do not want to evolve rules which obtain the highest rewards possible *in any state*, but to evolve rules which obtain the highest rewards possible *in the states in which they act*. That is, rather than rules which are *globally* good at obtaining reward, we want rules which are *locally* good at obtaining reward. In other words, we want rules whose actions are correct in all states they match. What's more, each state must be covered by a correct rule because an LCS must know how to act in each state; that is, it must have a *policy*.

To encourage the evolution of consistently correct rules, rather than rules which are good on average, we can use techniques like fitness sharing. But, while such techniques may help, there remains a fundamental mismatch between using strength as fitness and the goal of evolving rules with consistently correct actions. The effect of fitness sharing, and in particular its ability to combat overgeneral rules, deserves further study.

5.3.2 Strength and Best Action Maps

To maximise rewards, a strength-based LCS needs a population of rules which advocates the correct action in each state. If, in each state, only the best action is advocated, the population constitutes a *best action map* (Kovacs, 2002). While a best action map is an ideal representation in the sense that it is minimal, it is still possible to maximise rewards when incorrect actions are also advocated, as long as they are not selected. This is what we hope for in practice.

Now let's return to the question of how to define overgenerality in a strength-based system. Instead of saying X is overgeneral if some Y is fitter (stronger), let's say it is overgeneral if some Y is more consistent with the goal of forming a best action map; that is, if Y is correct in more cases than X.[3] Notice that we're now speaking of the correctness of rules (not just the correctness of actions), and of their relative correctness in specific. Let's emphasise these ideas:

Consistently Correct Rule: One which advocates a correct action in every state it matches. More formally, a classifier c is consistently correct w.r.t. a function f iff:

$$\forall s \in \mathcal{S}(c) \;\; f(s) = c(s)$$

Consistently Incorrect Rule: One which advocates an incorrect action in every state it matches. That is, a classifier c is consistently incorrect w.r.t. a function f iff:

$$\forall s \in \mathcal{S}(c) \;\; f(s) \neq c(s)$$

[3] In reinforcement learning terms, we could say X is overgeneral if some Y is more consistent with the optimal policy.

Correctness of a Rule: The correctness of a rule is the proportion of states in which it advocates the correct action.

The degree of correctness of a classifier c, correctness(c), w.r.t. a function f is the ratio between the states which it classifies correctly and the total number of states:

$$|\mathcal{C}| \; / \; |\mathcal{S}(c)|$$

where

$$\mathcal{C} = \{s \in \mathcal{S}(c) \mid c(s) = f(s)\}$$

Overgeneral Rule: One which advocates a correct action in some states and an incorrect action in others (i.e., a rule which is neither consistently correct nor consistently incorrect).

A classifier c is inconsistent w.r.t. a function f iff:

$$0 < \text{correctness}(c) < 1$$

That is, a classifier is inconsistent if it is neither consistently correct nor consistently incorrect.

The notion of the relative correctness of a rule allows us to say a rule Y is *more* correct (and hence *less* overgeneral) than a rule X, even if neither is consistently correct.

Now let's reevaluate E and F from figure 2 to see how consistent they are with the goal of forming a best action map. Rule E matches both states and advocates a correct action in both. This is compatible with forming a best action map, so E is not overgeneral. Rule F also matches both states, but advocates an incorrect action in state 0, making F incompatible with the goal of forming a best action map. Because a superior rule (D) can be obtained by specialising F, F is overgeneral.

Notice that we've now defined overgeneral rules twice: once in §5.3 and again above. For the tasks we're considering here the two definitions coincide, although this is not always true. For example, in the presence of perceptual aliasing (where an input to the LCS does not always describe a unique task state) a rule may be overgeneral by one definition but not by the other. That is, it may be neither consistently correct nor consistently incorrect, and yet it may be impossible to generate a more correct rule because a finer distinction of states cannot be expressed.

The above assumes the states referred to in the definition of overgenerality are task states. If we consider perceptual states rather than task states the rule is sometimes correct and sometimes incorrect *in the same state* (which is not possible in the basic tasks studied here). We could take this to mean the rule is not consistently correct, and thus overgeneral, or we might choose to do otherwise.

5.3.3 Accuracy and Complete Maps

While all reinforcement learners seek to maximise rewards, the approach of XCS differs from that of strength-based LCS. Where strength LCS seek to form best action maps, XCS seeks to form a *complete map*: a set of rules such that each action in each state is advocated by at least one rule (Wilson, 1995; Kovacs, 2000). This set of rules allows XCS to approximate the entire reward function and (hopefully) accurately predict the reward for any action in any state. XCS's fitness metric *is* consistent with this goal, and we'll use it to define the superiority of rules for XCS.

The different approaches to fitness mean that in strength-based systems we contrast consistently correct, consistently incorrect and overgeneral rules, but with accuracy-based fitness we contrast accurate and inaccurate rules.

In XCS, fitter rules are those with lower prediction errors – at least up to a point: small errors in prediction are ignored, and rules with small enough errors are considered fully accurate (specifically, those with error less than ε_0). In other words, XCS has some tolerance for prediction error, or, put another way, some tolerance for changes in a rule's strength, since changes in strength are what produce prediction error. This tolerance for prediction error is used to define overgenerality in XCS; we say that a rule is overgeneral if its prediction error exceeds the tolerance threshold, i.e., if $\varepsilon_j \geq \varepsilon_o$. In XCS 'overgeneral' is synonymous with 'not-fully-accurate'.

Although this work uses XCS as a model, we hope it will apply to other future accuracy-based LCS. To keep the discussion more general, instead of focusing on XCS and its error threshold, we'll refer to a somewhat abstract notion of tolerance called τ. Let $\tau \geq 0$ be an accuracy-based LCS's tolerance for oscillations in strength, above which a rule is judged overgeneral.

Like XCS's error threshold, τ is an adjustable parameter of the system. This means that in an accuracy-based system, whether a rule is overgeneral or not depends on how we set τ. If τ is set very high, then both E and F from figure 2 will fall within the tolerance for error and neither will be overgeneral. If we gradually decrease τ, however, we will reach a point where E is overgeneral while F is not. Notice that this last case is the reverse of the situation we had in §5.3.2 when using strength-based fitness. So which rule is overgeneral depends on our fitness metric.

5.3.4 Defining Overgenerality

To match the different goals of the two systems we need two definitions of overgenerality:

Strength-based overgeneral: For strength-based fitness, an overgeneral rule is one which matches multiple states and acts incorrectly in some, but not all.[4] That is, a rule c is a strength-based overgeneral w.r.t. a function f iff:

$$|\mathcal{S}(c)| > 1$$

$$\text{and } 0 < \text{correctness}(c) < 1$$

Accuracy-based overgeneral: For accuracy-based fitness, an overgeneral rule is one which matches multiple states, some of which return (sufficiently) different rewards, and hence has (sufficiently) oscillating strength. Here a rule is overgeneral if its oscillations exceed τ. That is, a rule c is an accuracy-based overgeneral w.r.t. a function f iff:

$$|\mathcal{S}(c)| > 1$$

$$\text{and } \varepsilon_c \geq \varepsilon_o$$

Note that the strength definition requires action on the part of the classifiers while the accuracy definition does not. Thus we can have overgenerals in a task which allows 0 actions (or, equivalently, 1 action) using accuracy (see, e.g., figure 5), but not using strength.

[4] This restatement of strength-based overgenerality is consistent with the two earlier definitions given in §5.3 and §5.3.2.

5.4 Strong Overgeneral Rules

Now that we've finally defined overgenerality satisfactorily let's turn to the subject of strong overgenerality. Strength is used to determine a rule's influence in action selection, and action selection is a competition between alternatives. Consequently it makes no sense to speak of the strength of a rule in isolation. Put another way, strength is a way of ordering rules. With a single rule there are no alternative orderings, and hence no need for strength. Therefore, for a rule to be a strong overgeneral, it must be stronger than another rule. In particular, a rule's strength is relevant when compared to another rule with which it competes for action selection.

Now we can define strong overgeneral rules, although to do so we need two definitions to match our two definitions of overgenerality:

Strength-based strong overgeneral: A rule which sometimes advocates an incorrect action, and yet whose *expected strength* is greater than that of some correct (i.e., not-overgeneral) competitor *for action selection*. That is, a rule c is a strength-based strong overgeneral w.r.t. a function f iff:

$$|\mathcal{S}(c)| > 1$$

$$\text{and } 0 < \text{correctness}(c) < 1$$

$$\text{and there is a consistently correct rule } r \text{ with strength}(r) < \text{strength(c)}$$

$$\text{with which } c \text{ competes for action selection.}$$

Accuracy-based strong overgeneral: A rule whose strength oscillates unacceptably, and yet whose *expected strength* is greater than that of some accurate (i.e., not-overgeneral) competitor *for action selection*. That is, a rule c is an accuracy-based strong overgeneral w.r.t. a function f iff:

$$|\mathcal{S}(c)| > 1$$

$$\text{and } \varepsilon_c \geq \varepsilon_o$$

$$\text{and there is a consistently correct rule } r \text{ with strength}(r) < \text{strength(c)}$$

$$\text{with which } c \text{ competes for action selection.}$$

The intention is that competitors be possible, not that they need actually exist in a given population.

The strength-based definition refers to competition with *correct* rules because strength-based systems are not interested in maintaining incorrect rules (see §5.3.2). This definition suits the analysis in this work. However, situations in which more overgeneral rules have higher fitness than less overgeneral – but still overgeneral – competitors are also pathological. Parallel scenarios exist for accuracy-based fitness. Such cases resemble the well-known idea of *deception* in genetic algorithms, in which search is led away from desired solutions (see, e.g., (Goldberg, 1989)).

5.5 Fit Overgeneral Rules

In our definitions of strong overgenerals we refer to competition for action selection, but rules also compete for reproduction. To deal with the latter case we introduce the concept of *fit overgenerals* as a parallel to that of strong overgenerals. A rule can be both, or either. The definitions for strength and accuracy-based fit overgenerals are identical to those for strong overgenerals, except that we refer to fitness (not expected strength) and competition for reproduction (not action selection):

Strength-based fit overgeneral: A rule which sometimes advocates an incorrect action, and yet whose *expected fitness* is greater than that of some correct (i.e., not-overgeneral) competitor *for reproduction*.[5]

Accuracy-based fit overgeneral: A rule whose strength oscillates unacceptably, and yet whose *expected fitness* is greater than that of some accurate (i.e., not-overgeneral) competitor *for reproduction*.

We won't consider fit overgenerals as a separate case in our initial analysis since in SB–XCS fitness and strength are the same, and so strong and fit overgenerals are similar.[6] Later, in §9, we'll see how XCS handles both fit and strong overgenerals.

5.6 Parallel Definitions of Strength and Fitness

At this point we must note two terminological issues. In strength-based systems, a rule's strength is used in action selection and reproduction, and so it is with SB–XCS.[7] Although XCS uses the same update as SB–XCS, Wilson refers to this value as *prediction*, rather than strength. For simplicity this value is generally referred to simply as strength in this work.

The term "strength", however, really has two interpretations. One is the value updated by (1), and the other is the weight a rule has in action selection. When aggregated over the relevant rules, the latter value is referred to as system strength in SB–XCS (see below) and system prediction (equation 7) in XCS.

In SB–XCS, the two notions of strength coincide; in SB–XCS a rule's contribution to the system strength for an action is just its strength weighted by numerosity (8). In XCS, however, a rule's contribution to the system prediction for an action is a function of both its prediction and fitness (7).

Consequently, in XCS we have two notions of strength; prediction, and contribution to system prediction. In discussing strong overgenerals, it should be understood that references to strength are to the latter value.

A similar problem occurs in referring to the fitness of a rule. In XCS, fitness is a value updated by (5). Fitness, however, can also be interpreted (in fact, is normally interpreted in evolutionary computation) as the weight of a rule in reproduction. If we consider a single invocation of the niche GA, a rule's fitness parameter coincides with its weight in reproduction. However, if we consider all invocations of the GA, a rule's weight in reproduction is partly determined by its generality, thanks to the niche GA (in both XCS and SB–XCS).

[5] The more formal definitions of fit overgenerals are omitted as they differ from those for strong overgenerals only in that they refer to fitness and reproduction instead of strength and action selection.

[6] Nonetheless, there is still a difference between strong and fit overgenerals in strength-based systems, since the two forms of competition may take place between different sets of rules. See §6.

[7] Strength-based systems may, however, distinguish between shared and unshared strength and use them differently.

In discussing fit overgenerals, it should be understood that references to fitness are to the weight of a rule in reproduction, not to the fitness parameter of a rule.

6 When are Strong and Fit Overgenerals Possible?

We've seen definitions for strong and fit overgeneral rules, but what are the exact conditions under which a task can be expected to produce them? If such rules are a serious problem for classifier systems, knowing when to expect them should be a major concern: if we know what kinds of tasks are likely to produce them (and how many) we'll know something about what kinds of tasks should be difficult for classifier systems (and how difficult).

Not surprisingly, the requirements for the production of strong and fit overgenerals depend on which definition we adopt. Looking at the accuracy-based definition of strong overgenerality we can see that we need two rules (a strong overgeneral and a not-overgeneral rule), that the two rules must compete for action selection, and that the overgeneral rule must be stronger than the not-overgeneral rule. The task conditions which make this situation possible are as follows:

1. The task must contain at least two states, in order that we can have a rule which generalises (incorrectly).[8]

2. The task may allow any number of actions in the two states, including 0 actions, or, equivalently, 1 action. (We'll see later that strength-based systems differ in this respect.)

3. In order to be a strong overgeneral, the overgeneral must have higher expected strength than the not-overgeneral rule. For this to be the case the reward function must return different values for the two rules. More specifically, it must return more reward to the overgeneral rule.

4. The overgeneral and not-overgeneral rules must compete for action selection. This constrains which tasks will support strong overgenerals.

The conditions which will support fit overgenerals are clearly very similar: 1) and 2) are the same, while for 3) the overgeneral must have greater fitness (rather than strength) than the not-overgeneral, and for 4) they must compete for reproduction rather than action selection.

6.1 The Reward Function is Relevant

Let's look at the last two requirements for strong overgenerals in more detail. First, in order to have differences in the expectations of the strengths of rules there must be differences in the rewards returned from the task. So the values in the reward function are relevant to the formation of strong overgenerals. More specifically, it must be the rewards returned to competing classifiers which differ. So subsets of the reward function are relevant to the formation of individual strong or fit overgenerals.

[8] We assume the use of the standard LCS language in which generalisation over actions does not occur. Otherwise, it would be possible to produce an overgeneral in a task with only a single state (and multiple actions) by generalising over actions instead of states.

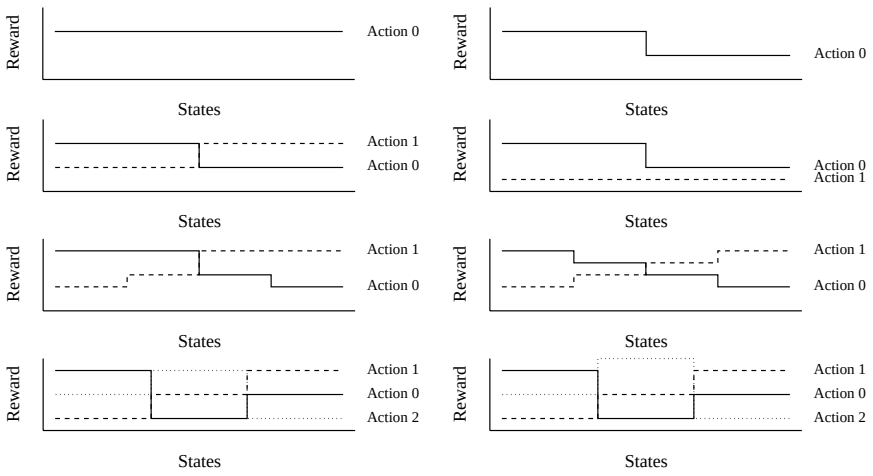


Figure 3: Reward functions with 1, 2 or 3 actions. Those on the left are unbiased while those on the right are biased.

Let us refer to the situation where different correct actions receive different rewards as a *bias* in the reward function. Reward functions which have no such biases are *unbiased*. More formally:

Unbiased reward function: A reward function R is unbiased iff there exists a constant r such that:

$$\forall s, a \text{ if correct}(s, a) \text{ then } R(s, a) = r$$

Figure 3 shows examples of unbiased reward functions on the left, and biased reward functions on the right. Note that the defining feature of the unbiased functions is that the highest reward in all states is the same constant value. If we plot the maximum reward for each state for any unbiased reward function we obtain a flat line.

For strong or fit overgenerals to occur, there must be a bias in the reward function at state-action pairs which map to competing classifiers. In the following section, we look at how classifiers can compete.

6.2 Competition for Action Selection

In XCS and SB–XCS, two classifiers c_1 and c_2 compete for action selection iff:

$$\mathcal{S}(c_1) \cap \mathcal{S}(c_2) \neq \emptyset \text{ and } \mathcal{A}(c_1) \neq \mathcal{A}(c_2)$$

Note that this relies on the property of the ternary language that classifiers are constant partial functions, i.e., that they advocate the same action in all states they match.

Figure 4 shows a reward function as a matrix with task state indexing the rows and LCS action indexing the columns. Using the standard ternary language rules can generalise over states but not actions, so a given rule will match some subset of a single column.

Rules compete for action selection when they occur in the same match set [M]. In XCS (and some other LCS) rules advocating the same action cooperate to have their

State \ Action			
	$R_{0,0}$	$R_{0,1}$	$R_{0,2}$
	$R_{1,0}$	$R_{1,1}$	$R_{1,2}$
	$R_{2,0}$	$R_{2,1}$	$R_{2,2}$

Figure 4: A reward function as a matrix indexed by state and action.

action chosen, while rules advocating different actions compete. (We could say competition is between action sets [A]s, rather than individual rules.) In other LCS, rules which advocate the same action also compete against each other, and only the winner receives reward.

In either case, two rules which occur in different action sets within a match set will compete. The rules in each action set are updated toward the rewards returned for different actions in a state (rows in the matrix). Hence differences within a row of the matrix influence the strengths of competing classifiers and may result in strong overgenerals.

The rewards returned for taking the same action in different states (columns in the matrix) also affect the strengths of competing rules simply because rules can generalise over multiple states, and their strengths depend on the values in all of them. Hence differences within columns can result in strong overgenerals.

The effect, of course, of strong overgenerals on competition for action selection is that the system will tend to select an incorrect action.

6.3 Competition for Reproduction

The locus of competition for reproduction depends on the GA scheme used. With a panmictic GA all rules in the population compete for reproduction, whereas with a niche GA only a subset of the population is eligible for reproduction. One way of running the niche GA is to restrict selection of rules to those in [M], in which case competition is between members of [M]. Similarly, with a niche GA in [A], rules in [A] compete.

More formally, using a niche GA in [M] two classifiers c_1 and c_2 compete for reproduction iff:

$$\mathcal{S}(c_1) \cap \mathcal{S}(c_2) \neq \emptyset$$

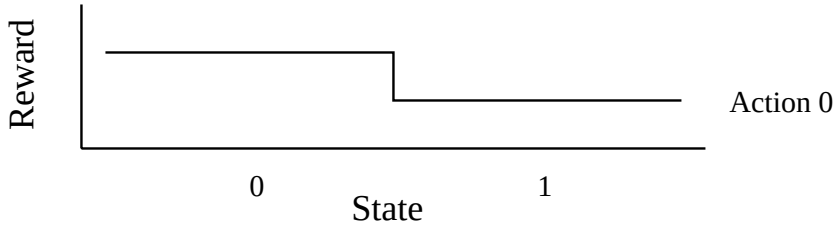
while using a niche GA in [A] c_1 and c_2 compete for reproduction iff:

$$\mathcal{S}(c_1) \cap \mathcal{S}(c_2) \neq \emptyset \text{ and } \mathcal{A}(c_1) = \mathcal{A}(c_2)$$

A niche GA limits the areas of competition within a population, and so limits how the rewards defined in the reward function interact. For example, with a GA in [A], rules belonging to different [A]s within an [M] do not compete for reproduction, so differences in the reward function between one [A] and another (i.e., differences within a row in the matrix) will not affect the reproduction of fit overgenerals in either.

Note, however, that different [A]s will overlap, as will different [M]s, and that a fully generalised rule is a member of all [M]s and all [A]s for its action. Even with a niche GA in [A] a fully general rule competes with all rules in the population which advocate the action it does. So even though a niche GA limits competition to subsets of the population, differences anywhere in a column can contribute to the fitness of two competing rules.

State	Action	Reward
0	0	$a = 1000$
1	0	$c = 0$



Rule	Condition	Action	E[Strength]
A	0	0	$a = 1000$
C	1	0	$c = 0$
E	#	0	$(a + c)/2 = 500$

Figure 5: A minimal (2x1) strong overgeneral task for XCS and all its classifiers.

This means that with a GA in [A] differences in the rewards given for the same action may contribute to the fitness of rules competing for reproduction, but differences in rewards for different actions do not. With a panmictic GA all rules compete, meaning a difference between any two parts of the reward function can contribute to the fitness of rules competing for reproduction. The effect, of course, of fit overgenerals on competition for reproduction is that the fit overgenerals will tend to propagate in the population.

7 Strong Overgenerals in XCS

In §6 we saw that, using the accuracy definition, strong overgenerals require a task with at least two states, and that each state can have any number of actions. We also saw that the reward function was relevant but did not see exactly how. Now let's look at a minimal strong overgeneral supporting task for accuracy and see exactly what is required of the reward function to produce strong overgenerals. Figure 5 shows a reward function for a task with two states and one action and all possible classifiers for it. As always, the expected strengths shown are due to the simplifying assumption that states and actions occur equiprobably (§4).

In §4 we made a number of simplifying assumptions, and for now let's make a further one: that there is no tolerance for oscillating strengths ($\tau = 0$), so that any rule whose strength oscillates at all is overgeneral. This means rule E in figure 5 is an overgeneral because it is updated toward different rewards. It is also a strong overgeneral because it is stronger than some not-overgeneral rule with which it competes, namely rule C. In §6 we saw that strong overgenerals depend on the reward function returning more strength to the strong overgeneral than its not-overgeneral competitor. Are there reward functions under which E will not be a strong overgeneral? Since the strength of E is an average of the two rewards returned (labelled a and c in figure 5 to correspond to the names of the fully specific rules which obtain them), and the strength of C is c, then as long as $a > c$, rule E will be a strong overgeneral. Symmetrically, if $c > a$ then E will still be a strong overgeneral, in this case stronger than not-overgeneral rule A.

State	Action	Reward
0	0	w
1	0	x

State	Action	Reward
0	1	y
1	1	z

Rule	Condition	Action	E[Strength]	Overgeneral unless
A	0	0	w	never
B	0	1	y	never
C	1	0	x	never
D	1	1	z	never
E	#	0	$(w+x)/2$	$\|w-x\| \leq \tau$
F	#	1	$(y+z)/2$	$\|y-z\| \leq \tau$

Figure 6: A task which demonstrates that biases between actions do not result in strong overgenerals.

The only reward functions which do not cause strong overgenerals are those in which $a = c$. So in this case *any* bias in the reward function makes the formation of a strong overgeneral possible.

If we allow some tolerance for oscillations in strength without judging a rule overgeneral, then rule E is not overgeneral only if $a - c \leq \tau$ where τ is the tolerance for oscillations. In this case only reward functions in which $a - c > \tau$ will produce strong overgeneral rules.

7.1 Biases between Actions do not Produce Strong Overgenerals

We've just seen an example where any bias in the reward function will produce strong overgenerals. However, this is not the case when we have more than one action available, as in figure 6. The strengths of the two fully generalised rules, E & F, are dependent only on the values associated with the actions they advocate. Differences in the rewards returned for different actions do not result in strong overgenerals – as long as we don't generalise over actions, which was one of our assumptions from §4.

7.2 Some Properties of Accuracy-Based Fitness

At this point it seems natural to ask how common strong overgenerals are and, given that the structure of the reward function is related to their occurrence, to ask what reward functions make them impossible. In this section we'll prove some simple, but perhaps surprising, theorems concerning overgeneral rules and accuracy-based fitness. However, we won't go too deeply into the subject for reasons which will become clear in §9.

Let's begin by looking at a special kind of reward function in which strong overgenerals are impossible, and proving this is so. Immediately after we'll see the general conditions which make strong overgenerals impossible, but, first, the special case:

Theorem 1 *In XCS, overgeneral rules are impossible when the reward function is constant over each action.*

Proof. The strength of a rule is a function of the values it updates toward, and these values are a subset of the rewards for the advocated action. If all such rewards are equivalent there can be no oscillations in a rule's strength, so it cannot be overgeneral. □

More generally, strong overgenerals are impossible when the reward function is sufficiently close to constancy over each action that oscillations in any rule's strength are less than τ. Now we can see when strong overgenerals are possible:

Theorem 2 *In XCS, if the task structure meets requirements 1 and 4 of §6 at least one overgeneral rule will be possible for each action for which the reward function is not within τ of being constant.*

Proof. A fully generalised rule matches all inputs and its strength is updated toward all possible rewards for the action it advocates. Unless all such rewards are within τ of equivalence it will be overgeneral. □

In other words, if the rewards for the same action differ by more than τ the fully generalised rule for that action will be overgeneral. To avoid overgeneral rules completely, we'd have to constrain the reward function to be within τ of constancy for each action. That overgeneral rules are widely possible should not be surprising. But it turns out that with accuracy-based fitness there is no distinction between overgeneral and strong overgeneral rules:

Theorem 3 *In XCS, all overgeneral rules are strong overgenerals.*

Proof. Let's consider the reward function as a vector $R = [r_1\ r_2\ r_3\ \ldots\ r_n]$, and, for simplicity, assume $\tau = 0$. An overgeneral matches at least two states, and so is updated toward two or more distinct values from the vector, whereas accurate rules are updated toward only one value (by definition, since $\tau = 0$) no matter how many states they match. For each r_i in the vector there is some fully specific (and so not overgeneral) rule which is only updated toward it. Consequently, any overgeneral rule (which must match at least two states) competes with at least two accurate rules.

Now consider the vector $X = [x_1\ x_2\ x_3\ \ldots\ x_y]$ which is composed of the subset of vector R toward which the overgeneral in question is updated. Because we've assumed states and actions occur equiprobably, the strength of a rule is just the mean of the values it is updated toward. So the strength of the overgeneral is $\bar{X}$, the mean of X.

The overgeneral will be a strong overgeneral if it is stronger than some accurate rule with which it competes. The weakest strength for such a rule is $\min x_i$. The inequality $\min x_i < \bar{X}$ is true for all reward vectors except those which are constant functions, so all overgenerals are strong overgenerals. □

Taking theorems 2 and 3 together yields:

Theorem 4 *In XCS, if the task structure meets requirements 1 and 4 of §6 at least one* **strong** *overgeneral rule will be possible for each action for which the reward function is not within τ of being constant.*

In short, using accuracy-based fitness and reasonably small τ only a highly restricted class of reward functions and tasks do not support strong overgeneral rules.

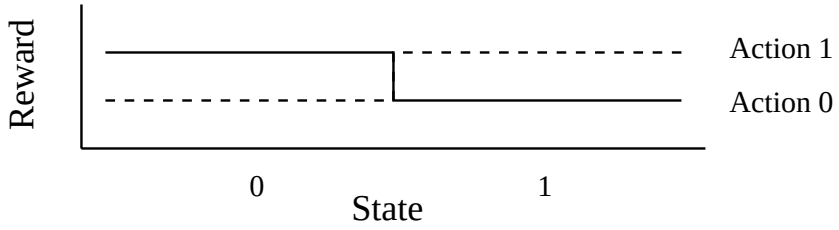
These 4 theorems are independent of the number of actions available in a task. Note that the 'for each action' part of the theorems depends on the inability of rules to generalise over actions, a syntactic limitation of the standard LCS language. If we remove this arbitrary limitation then we further restrict the class of reward functions which will not support strong overgenerals.

8 Strong Overgenerals in SB–XCS

We've seen how the reward function determines when strong overgeneral classifiers are possible in accuracy-based systems. Now let's look at the effect of the reward function using SB–XCS, our strength-based system. Recall from the strength-based definition of

State	Action	Reward
0	0	1000
1	0	0

State	Action	Reward
0	1	0
1	1	1000



Rule	Condition	Action	E[Strength]	Status
A	0	0	1000	Correct
B	0	1	0	Incorrect
C	1	0	0	Incorrect
D	1	1	1000	Correct
E	#	0	500	Overgeneral
F	#	1	500	Overgeneral

Figure 7: An unbiased reward function and all its classifiers. Unbiased functions will not cause strong overgenerals.

strong overgenerals that we need two rules (a strong overgeneral and a not-overgeneral correct rule), that the two rules must compete for action selection, and that the overgeneral rule must be stronger than the correct rule. The conditions which make this situation possible are the same as those for accuracy-based systems, except for a change to condition 2: there needs to be at least one state in which at least two actions are possible, so that the overgeneral rule can act incorrectly. (It doesn't make sense to speak of overgeneral rules in a strength-based system unless there is more than one action available.)

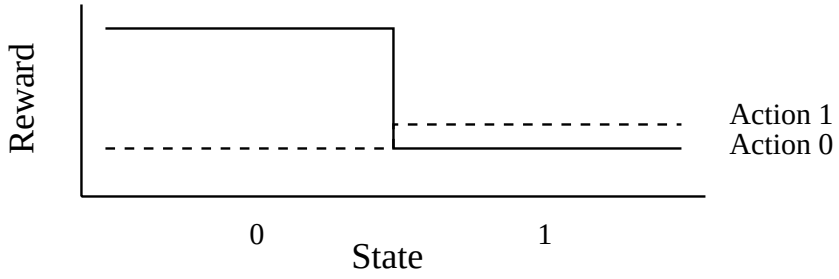
A second difference is that in strength-based systems there is no tolerance for oscillations in a rule's strength built into the update rules. This tolerance is simply not needed in SB–XCS where all that matters is that a rule advocate the correct action, not that its strength be consistent.

A complication to the analysis done earlier for accuracy-based systems is that strength-based systems tend toward best action maps (§5.3.2). Simply put, SB–XCS is not interested in maintaining incorrect rules, so we are interested in overgenerals only when they are stronger than some correct rule. For example, consider the binary state binary action task of figure 7. Using this unbiased reward function, rules E & F are overgenerals (since they are sometimes incorrect), but not strong overgenerals because the rules they are stronger than (B & C) are incorrect. (Recall from the definition of a strong overgeneral in a strength LCS in §5.4 that the strong overgeneral must be stronger than a *correct* rule.) This demonstrates that in strength-based systems (unlike accuracy-based systems) not all overgeneral rules are strong overgenerals.

What consequence does this disinterest in incorrect rules have on the dependence of strong overgenerals on the reward function? The reward function in this example is not constant over either action, and the accuracy-based concept of tolerance does

State	Action	Reward
0	0	$w = 1000$
1	0	$x = 0$

State	Action	Reward
0	1	$y = 0$
1	1	$z = 200$



Rule	Condition	Action	E[Strength]	Strong overgeneral if
A	0	0	$w = 1000$	never
B	0	1	$y = 0$	never
C	1	0	$x = 0$	never
D	1	1	$z = 200$	never
E	#	0	$(w + x)/2 = 500$	$(w + x)/2 > z$
F	#	1	$(y + z)/2 = 100$	$(y + z)/2 > z$

Figure 8: A 2x2 biased reward function which is a minimal strong overgeneral task for strength-based LCS, and all its classifiers.

not apply. In an accuracy-based system there must be strong overgenerals under such conditions, and yet there are none here.

8.1 When are Strong Overgenerals Impossible in SB–XCS?

Let's begin with a first approximation to when strong overgenerals are impossible. Later, in §8.2, we'll ask when strong overgenerals *are* possible, and we'll get a more precise answer to the question of when they are impossible.

Theorem 5 *In SB–XCS, strong overgenerals are impossible when the reward function is unbiased (i.e., constant over correct actions).*

Proof. A correct action is one which receives the highest reward possible in its state. If all correct actions receive the same reward, this reward is higher than that for acting incorrectly in *any* state. Consequently no overgeneral rule can have higher strength than a correct rule, so no overgeneral can be a strong overgeneral. □

To make theorem 5 more concrete, reconsider the reward values in figure 6. By definition, a correct action in a state is one which returns the highest reward for that state, so if we want the actions associated with w and z to be the only correct actions then $w > y, z > x$. If the reward function returns the same value for all correct actions then $w = z$. Then the strengths of the overgeneral rules are less than those of the correct accurate rules: E's expected strength is $(w+x)/2$ which is less than A's expected strength of w and F's expected strength is $(y + z)/2$ which is less than D's z, so the overgenerals cannot be strong overgenerals. (If $w < y$ and $z < x$ then we have a symmetrical situation in which the correct action is different, but strong overgenerals are still impossible.)

8.2 What Makes Strong Overgenerals Possible in SB–XCS?

It is possible to obtain strong overgenerals in SB–XCS by defining a reward function which returns different values for correct actions. An example of a minimal strong overgeneral supporting task for SB–XCS is given in figure 8. Using this reward function, E *is* a strong overgeneral, as it is stronger than the correct rule D with which it competes for action selection (and for reproduction if the GA runs in the match set or panmictically).

However, not all differences in rewards are sufficient to produce strong overgenerals. How much tolerance does SB–XCS have before biases in the reward function produce strong overgenerals? Suppose the rewards are such that the actions associated with w and z are correct (i.e., $w > y, z > x$) and the reward function is biased such that $w > z$. How much of a bias is needed to produce a strong overgeneral? That is, how much greater than z must w be? Rule E competes with D for action selection, and will be a strong overgeneral if its expected strength exceeds D's, i.e., if $(w+x)/2 > z$, which is equivalent to $w > 2z - x$. So a bias of $w > 2z - x$ means E will be a strong overgeneral with respect to D, while a lesser bias means it will not.

E also competes with A for reproduction, and will be fitter than A if $(w+x)/2 > w$, which is equivalent to $x > w$. So a bias of $x > w$ means E will be a fit overgeneral with respect to A, while a lesser bias means it will not. (Symmetrical competitions occur between F & A (for action selection) and F & D (for reproduction).)

We'll take the last two examples as proof of the following theorem:

Theorem 6 *In SB–XCS, if the task structure meets requirements 1 and 4 of §6 and the modified requirement 2 from §8, a strong overgeneral is possible whenever the reward function is biased such that* $(w + x)/2 > z$ *for any given rewards* w, x *&* z.

8.3 SB–XCS's Tolerance for Reward Biases

The examples in the previous section show there is a certain tolerance for biases (differences) in rewards within which overgenerals are not strong enough to outcompete correct rules. Knowing what tolerance there is is important as it allows us to design reward functions which will not produce strong overgenerals. Unfortunately, because of the simplifying assumptions we've made (see §4) these results do not apply to more realistic tasks. However, they do tell us how biases in the reward function affect the formation of strong overgenerals, and give us a sense of the magnitudes involved. An extension of this work would be to find limits to tolerable reward function bias empirically. Two results which do transfer to more realistic cases are theorems 1 and 5, which tell us under what conditions strong overgenerals are impossible for the two types of LCS. These results hold even when our simplifying assumptions do not.

9 Fit Overgenerals and the Survival of Rules under the GA

We've examined the conditions under which strong overgenerals are possible under both types of fitness. The whole notion of a strong overgeneral is that of an overgeneral rule which can outcompete other, preferable, rules. But, as noted earlier, there are two forms of competition: action selection and reproduction. Our two systems handle the first in the same way, but handle reproduction differently. In this section we examine the effect of the fitness metric on the survival of strong overgenerals.

XCS and SB–XCS were compared empirically on the tasks in figures 7 and 8. The GA was disabled and all possible rules inserted in the LCS at the outset. Settings were $\beta = 0.2$, and $\varepsilon_o = 0.01$. Wilson's pure explore / exploit scheme (Wilson, 1995) was used.

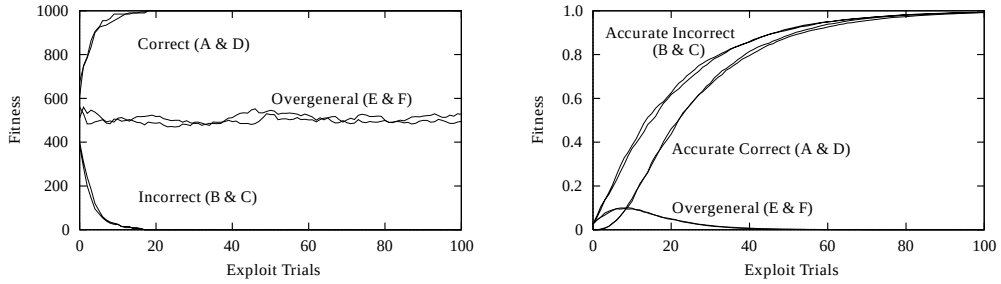


Figure 9: Rule fitness using strength-based SB–XCS (left) and accuracy-based XCS (right) on the unbiased function from figure 7.

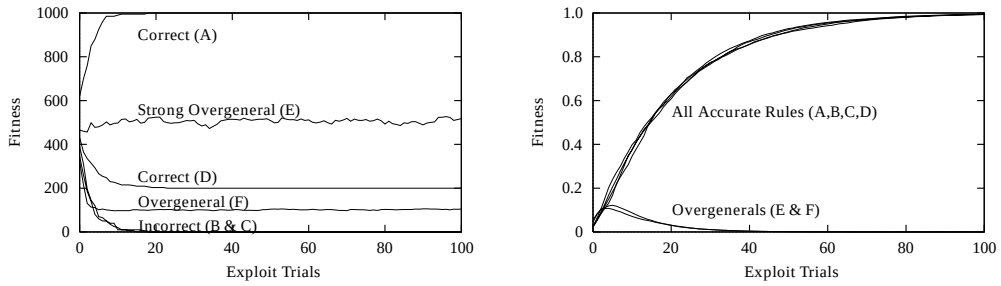


Figure 10: SB–XCS (left) and XCS (right) on the biased function from figure 8.

9.1 Comparison on an Unbiased Reward Function

First we compared XCS and SB–XCS on the reward function from figure 7. Figure 9 shows the fitness of each rule using strength (left) and accuracy (right), with results averaged over 100 runs. The first thing to note is that we are now considering the development of a rule's strength and fitness over time (admittedly with the GA turned off), whereas until this section we had only considered steady state strengths (as pointed out in §4). We can see that the actual strengths indeed converge toward the expected strengths shown in figure 7. We can also see that the strengths of the overgeneral rules (E & F) oscillate as they are updated toward different values.

Using strength (figure 9, left), the correct rules A & D have highest fitness, so if the GA was operating we'd expect SB–XCS to reproduce them preferentially and learn to act correctly in this task.

Using accuracy (figure 9, right), all accurate rules (A, B, C & D) have high fitness, while the overgenerals (E & F) have low fitness. Note that even though the incorrect rules (B & C) have high fitness and will survive with the GA operational, they have low strength, so they will not have much influence in action selection. Consequently we can expect XCS to learn to act correctly in this task.

9.2 Comparison on a Biased Reward Function

While both systems seem to be able to handle the unbiased reward function, compare them on the same task when the reward function is biased as in figure 8. Consider the results shown in figure 10 (again, averaged over 100 runs). Although XCS (right)

treats the rules in the same way now that the reward function is biased, SB–XCS (left) treats them differently. In particular, rule E, which is overgeneral, has higher expected strength than rule D, which is correct, and with which it competes for action selection. Consequently E is a strong overgeneral (and a fit overgeneral if E and D also compete for reproduction). (Further notes on figures 9 and 10 are available in (Kovacs, 2002).)

9.3 Discussion

In these trivial tasks XCS's accuracy-based fitness is effective at penalising overgeneral, strong overgeneral, and fit overgeneral rules. This shouldn't be surprising: for accuracy, we've defined overgeneral rules precisely as those which are less than fully accurate. With fitness based on accuracy these are precisely the rules which fare poorly.

With SB–XCS's use of strength as fitness, strong overgenerals are fit overgenerals. But with XCS's accuracy-based fitness, strong overgenerals – at least those encountered so far – have low fitness and can be expected to fare poorly.

10 Designing Strong and Fit Overgenerals for XCS

We have already examined the effect of bias in the reward function, and we have seen that it may produce strong overgenerals in both XCS and SB–XCS, and fit overgenerals in SB–XCS. However, we have also seen that biases in the reward function do not produce fit overgenerals in XCS, since overgenerals (including strong overgenerals) have low fitness under accuracy-based fitness.

That is, accuracy-based fitness gives XCS immunity to fit overgenerals caused by biases in the reward function. Is there a way in which we can induce strong and fit overgenerals in XCS? Perhaps there is another form of bias to which XCS is susceptible.

10.1 Biased Variance Functions

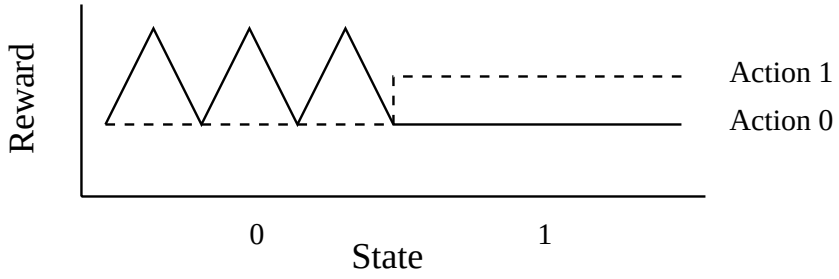
Just as we can bias the rewards themselves, so can we bias the variance in the rewards. That is, we can construct a reward function which has more variance in the reward for one state-action than another. This gives us what we might call a biased *variance function*. (We might also say it gives us a biased *accuracy function*. Suppose we applied tabular Q-learning to a task with a biased variance function and calculated the accuracy of each state-action using XCS's accuracy-based fitness updates. The accuracies of the state-action pairs give us a form of accuracy function, and it will be biased according to the bias in the variance function.)

It was hypothesised that XCS would suffer from strong and fit overgenerals if the variance in the reward function was suitably biased. For example, figure 11 shows a task with unbiased rewards, but vastly different variance (σ^2) in the rewards for different state-actions; while the other state-actions all have 0 variance, state-action 0:0 has variance 100. A sawtooth waveform is used in the appropriate part of the state / reward diagram in figure 11 in order to suggest variance in the reward for this state-action. Note that the rewards for all other state-actions are shown with flat lines to indicate they have 0 variance. An alternative way to show the variance for this reward function is to collapse the state-actions onto the x-axis and show the variance on the y-axis.

Because the variance in reward for state 0 action 0 is so high, rule A will have low fitness in XCS (unless the accuracy criterion ε_o is quite high). In fact, E and F, which are overgeneral, may have higher fitness than A, and be fit overgenerals with respect to A. They may also be strong overgenerals with respect to A; although they have less strength than A, the inclusion of fitness in the system prediction calculation (§7) might discount the prediction of A enough to give E and F more influence in action selection.

State	Action	Reward	σ^2
0	0	100	100
1	0	90	0

State	Action	Reward	σ^2
0	1	90	0
1	1	100	0



Rule	Condition	Action	E[Strength]	Fitness
A	0	0	100	low
B	0	1	90	high
C	1	0	90	high
D	1	1	100	high
E	#	0	95	medium
F	#	1	95	medium

Figure 11: A task in which XCS may suffer from fit overgenerals. The sawtooth wave indicates a state-action in which rewards are stochastic.

10.2 Empirical Results

In order to determine whether XCS would develop fit overgenerals in such conditions, XCS was evaluated in the environment in figure 11 with different levels of variance for state-action 0:0. For each level of variance, XCS was initialised with a fixed population consisting of the 6 possible rules in figure 11. At the start of each run the parameters of these rules were initialised to default values unrelated to the strength and fitness values shown there. Rule discovery was disabled so that XCS operated like a tabular Q-learner. Parameter settings were as follows: Subsumption threshold $\theta_{sub} = 20$, GA threshold $\theta_{GA} = 25$, t3 deletion threshold $\theta_{del} = 25$, Covering threshold $\theta_{mna} = 1$, Low-fitness deletion threshold $\delta = 0.1$, Population size limit $N = 400$, Learning rate $\beta = 0.2$, Accuracy falloff rate $\alpha = 0.1$, Accuracy criterion $\varepsilon_o = 0.01$, Crossover rate $\chi = 0.8$, Mutation rate $\mu = 0.04$, Hash probability $P_\# = 0.33$.

XCS was first tested using the rewards in figure 11 with no variance; that is, the rewards were deterministic. Next, the reward for state-action 0:0 was made stochastic, and its variance increased incrementally. For each test, the fitness of each of the 6 rules was recorded after 1000 time steps, which was enough time for them to converge to stable values. Figure 12 shows the fitness of rules A and E as the variance in the reward for 0:0 increased from test to test. This figure shows the variance in terms of the range of rewards for 0:0. In each test, rewards for 0:0 were obtained by generating a uniform random number r between 0 and R inclusive. The R used for various tests is shown on the x-axis of figure 12. The reward given to XCS for state-action 0:0 was $100 + r - 0.5R$ so that, for example, with $R = 20$, rewards were between 90 and 110, and with $R = 40$, rewards were between 80 and 120. As the range in rewards R increases from 30 to 40, the two curves cross over, and E becomes a fit overgeneral with respect to A.

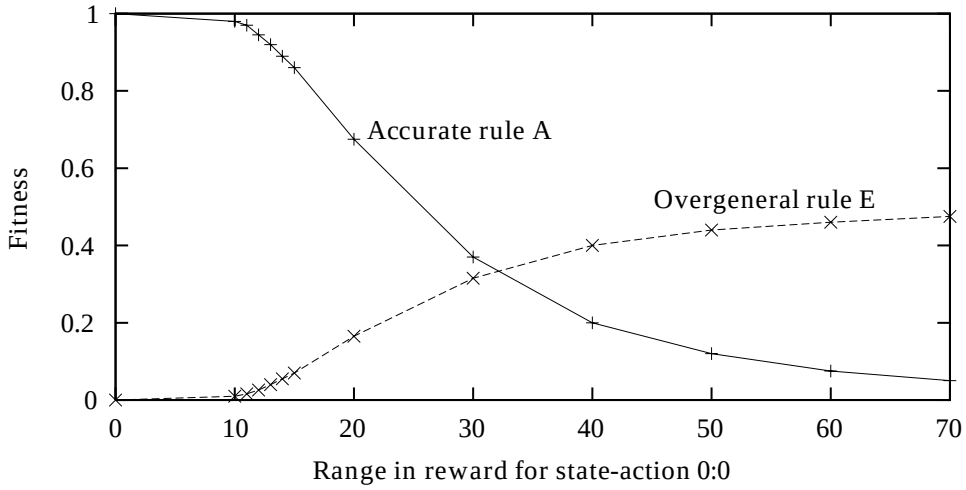


Figure 12: Fitness of two rules from figure 11 at different levels of variance bias. Rule E becomes a fit overgeneral in XCS when the variance in reward for state-action 0:0 is sufficiently high.

10.3 SB–XCS and Biased Variance Functions

How does SB–XCS fare with biased variance functions? Will it suffer from fit overgenerals (as XCS does) in the task in figure 11? Expected rule strength is not affected by the increase in variance, so SB–XCS should not be affected, other than by stochastic effects, e.g., long sequences of atypical rewards drawn from the uniform distribution, which will occasionally happen.

Thus, SB–XCS suffers from biases in the reward function while XCS suffers from biases in the variance function. Neither suffers from the other's problem. In other words, each is immune to the conditions which cause fit overgenerals in the other! The obvious question is: which problem is worse? Given the results we will see on the prevalence of value function bias in sequential tasks in §12, susceptibility to variance function bias may be the lesser of two evils, but further study is needed.

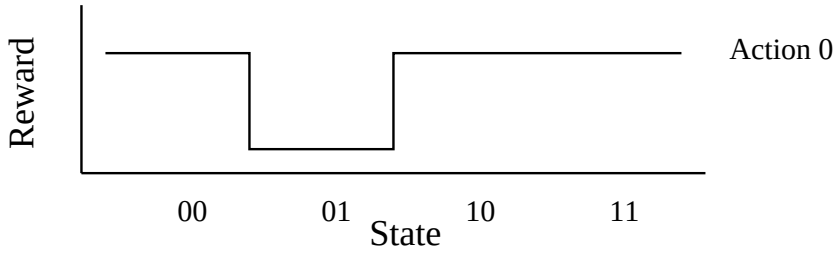
Note that SB–XCS's immunity to biases in the variance function gives us a way of constructing SB–XCS-easy yet XCS-hard functions. Another approach, based on the fact that SB–XCS tends not to represent lower-reward state-actions while XCS does represent them, is demonstrated in (Kovacs, 2002).

11 Strong and Fit Undergeneral Rules

We have seen strong and fit overgeneral rules, which are overgenerals that are stronger or fitter than some correct competitor, and that they result from biases in the reward or variance function. Now we will see that it is also possible for biases in the reward function to produce undergeneral rules which are stronger and fitter than a more-general-yet-correct competitor. The problem with this is that it interferes with the evolution of (accurate) general rules, and so accurate, general representations of the task at hand.

The task in figure 13 demonstrates such a case. At the bottom of the figure are three rules of particular interest, their expected strengths and an evaluation of their generality. Although rule C is the most general-yet-accurate, rules B and A have higher

State	Action	Reward
00	0	1000
01	0	200
10	0	1000
11	0	1000



Rule	Condition	Action	E[Strength]	Generality
A	00	0	1000	undergeneral
B	1#	0	1000	undergeneral
C	##	0	800	optimally general

Figure 13: A task which supports strong undergeneral rules.

strength. (The principle can be demonstrated with only 2 states, but the example seemed clearer with 4.) This is not a problem in action selection since they all advocate the same action, but it is in reproduction. That is, strong undergeneral rules are not a problem, but fit undergeneral rules are. Let's consider SB–XCS and XCS in turn.

11.1 SB–XCS

With SB–XCS's strength / fitness, the less general rules are fitter, which is clearly undesirable. This effect of fit undergenerals depends on the reward function being biased; with an unbiased reward function, fit undergenerals cannot occur. However, even with unbiased reward functions there must still be some bias toward generality, otherwise SB–XCS will not prefer the more general of two consistently correct rules. This is a real problem, since without a generality preference the population will swell up with an enormous number of correct but overly specific rules.

These problems demonstrate that SB–XCS must factor generality into rule fitness if it is to evolve accurate, general rules. Happily, the niche GA provides an effective fitness bonus which does just this.

11.2 XCS

In XCS, the fitness of A depends on the setting of the accuracy criterion. If it is strict enough, A is, by definition, an overgeneral, and the low fitness it receives is appropriate.

If, however, A is not overgeneral according to the accuracy criterion, it is only as fit as B and C. In other words, XCS will have equal preference for the three, on the basis of their fitness. Thus, although XCS does not suffer from fit overgenerals (unless the variance function is sufficiently biased §10), it still needs some bias toward generality, which, happily, it has thanks to the niche GA.

12 Sequential Tasks

In non-sequential tasks a reinforcement learner approximates the reward function defined by the experimenter, which is in principle enough to maximise the reward it receives. In sequential tasks, however, consideration of the reward function alone is not sufficient, as it defines immediate reward (the reward on a given time step) only. In sequential tasks the learner's actions influence the state of the task and hence which rewards it may receive in the future. Consequently the learner must take into account future consequences of current actions if it is to maximise the total amount of reward it receives over multiple time steps. Many reinforcement learning systems do so by learning a *value function*, which maps each state to an estimate of its long-term value. The value function for a task is implicitly defined by the task definition.

12.1 Q-learning

We'll examine the learning of sequential tasks by Q-learning agents, since XCS and SB–XCS both use the Q-update to estimate rule strength. Q-learners take long-term consequences into account by learning a *Q-function* (also called an *action-value* function) which maps state-action pairs to an estimate of their long term value called their *Q-value*. (Q stands for quality of the state-action pair.) A Q-function is really just a special kind of value function, in which we estimate the long-term value of state-action pairs rather than states. In fact, we can define a value function $V(s)$ as:

$$V(s) = \max_a Q(s,a) \tag{10}$$

That is, the value of a state s is the value of its highest Q-value.

In non-sequential tasks XCS and SB–XCS both update rules strengths toward their immediate reward, and the population of rules estimates the reward function. In sequential tasks, when using the sequential definition of P (equation 6), the strength of a rule is an estimate of long-term value, and the population of rules and their strengths approximate the value function. Just as in non-sequential tasks, strong and fit overgenerals depend on the reward function, so in sequential tasks they depend on the value function (or Q-function). Consequently, it is of interest to define a unbiased value functions, under which strong and fit overgenerals are impossible following theorems 1 and 5.

Unbiased value function: A value function V is unbiased iff there exists a constant c such that:

$$V(s) = c \tag{11}$$

for all $s \in \mathcal{S}$. That is, a value function is unbiased if it is a constant function.

Recall from §6.1 that a reward function is unbiased if it is constant *over correct actions*, not simply constant. The difference occurs because the value function is not parameterised by actions; by (10) the value of a state is the value of its highest-valued state-action.

12.2 The Need to Pass Values Back

Q-values are estimates of the long-term value of taking a given action in a given state, not just the immediate reward for doing so. To effect this, each Q-value is updated

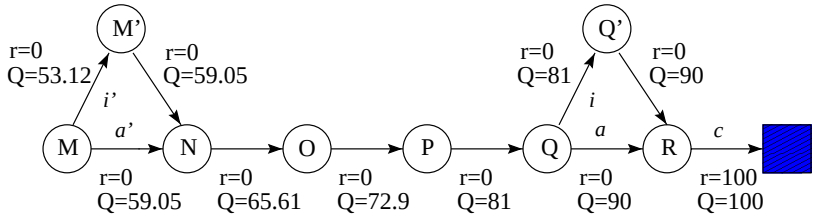


Figure 14: A simple sequential task showing immediate rewards r and Q-values Q for state transitions using $\gamma = 0.9$. M is the start state and the square state is terminal.

toward a fraction of the value of its successor. In this way, the estimate of the value of acting now comes to reflect some of the value of what happens later. To see how this works, let's look at the Q-update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \beta \underbrace{\left[\overbrace{r_{t+1} + \gamma \max_a Q(s_{t+1}, a)}^{Target} - \overbrace{Q(s_t, a_t)}^{OldEstimate} \right]}_{Error} \tag{12}$$

where $Q(s, a)$ is the quality (Q-value) of state-action pair s, a, t is the time step, β is the learning rate, and r is the immediate reward.

Q-values are updated toward the target component of (12), that is, toward the immediate reward r plus some fraction γ of the value of the state which follows it ($\max_a Q(s_{t+1}, a)$). (We say the value of the following state s_{t+1} is discounted and passed back to its predecessor s_t.) In this way a state-action which receives an immediate reward of 0 will have a Q-value greater than 0 if the state it leads to has a Q-value greater than 0 (assuming $\gamma > 0$). In other words, by passing value back a state takes on some of the value of its successor. In this way, Q-updates drive Q-values toward the long-term value of a state-action.

A simple task is shown in figure 14 to illustrate Q-learning. The immediate reward r and true Q-value Q for each state-action, assuming a discount rate of $\gamma = 0.9$, are shown. We define the value of the terminal state to be 0 so that the value of the transition labelled c is due entirely to the immediate reward there.

We can initialise estimates of each Q-value arbitrarily, and the Q-updates will move them toward the true Q-values shown. (In fact, given infinite revisits to each state-action, and appropriately declined α, the estimated Q-values are guaranteed to converge to their true values.)

Knowledge of the true (as opposed to estimated) Q-function for a task is sufficient to act optimally by simply taking the action in the current state with the highest associated Q-value. In figure 14 this means taking a' rather than i' and a rather than i.

12.3 The Need for Discounting

We've just seen that we must pass value back in order to take long-term consequences into account. But how much value should we pass back? In the Q-update, this is parameterised by the discount rate γ. We can think of γ controlling how much consideration the system gives to future rewards in making decisions. At γ = 1.0 no discounting occurs, and the system will learn the path through the task's states which results in the most reward, regardless of how long the path is. This is often not what we want.

For example, being paid £5 a year from now is not as desirable as being paid £5 today, for a number of reasons, e.g., the £5 cannot be spent until it has been paid a year from now, over the course of a year inflation will reduce its value, and, worse, either party might die within a year, preventing payment. If we used $\gamma = 1.0$ in figure 14 then both the i and a transitions would have Q-values of 100 and the system would be unable to choose between them. At the other extreme, if we set γ to 0.0 the system will be shortsighted and take no interest in the future consequences of its actions. This too is often undesirable, as it would lead the system to choose £5 today rather than £1000 tomorrow. In figure 14, $\gamma = 0.0$ would give i and a Q-values of 0, and again the system would be unable to choose between them. Typically we will want to set γ to some value between 0 and 1 in order to give possible future rewards a suitable weighting.

12.4 How Q-functions become Biased

Passing values back from one state to another is necessary for the system to take future consequences of its actions into account. Discounting of these values is necessary for the system to find shorter paths through the state space. However, passing values back and discounting tend to produce one of the two criteria for the production of strong overgenerals: a biased Q-function. Figure 14 demonstrates this effect. Notice that even though there are only two different immediate rewards (r is either 0 or 100), there are many Q-values. Even though the reward function has only two different rewards for correct actions (and so is a little biased) the Q function has many different rewards for correct actions (and so has more biases than the reward function).

The other criterion for the formation of strong overgenerals, that it be possible to act incorrectly, is met in all non-trivial tasks, specifically, in all tasks in which we have a choice of alternative actions in at least one state.

12.5 Examples

Let's look at some examples of strong overgenerals based on figure 14.

12.5.1 Short Sequences can Produce Strong Overgenerals

Imagine the situation where an overgeneral matches the state-actions labelled c and i. Now imagine a second rule which is correct and which advocates only the state-action labelled a. This rule competes with the overgeneral for action selection in state Q, and its strength is 90.

Using the strength definition of strong overgenerality (§5.4), the overgeneral rule is a strong overgeneral if it is stronger than a correct competitor. If we assume the overgeneral experiences both the transitions it advocates with equal frequency, we can use the inequality from theorem 6 to tell us whether it is a strong overgeneral:

$$(c + i)/2 > a \tag{13}$$

which evaluates to:

$$(100 + 81)/2 > 90$$
$$90.5 > 90$$

This example demonstrates that strong overgenerals can be obtained even with very short sequences of states, even the minimal sequence shown here.

12.5.2 Any $0 < \gamma < 1$ can Produce Strong Overgenerals

If the value of a state s is $V(s)$, discounting results in $\gamma V(s)$ being passed to the state preceding s. More generally, a state n steps ahead of s receives $\gamma^n V(s)$ from s. Let n be the number of steps between c and both i and a. Then we can rewrite the strong overgeneral inequality (13) as:

$$(c + \gamma^n c)/2 > \gamma^n c \tag{14}$$

which is true for any $c > 0$, $n \geq 1$, and, significantly, any $0 < \gamma < 1$. In other words, according to our approximate expression, passing back *any* fraction of the value of a state (other than all of it, i.e., $\gamma = 1$) will produce a Q-function capable of supporting strong overgeneral rules in this task. This does not occur if $\gamma = 1$, but we've already seen in §12.3 that passing back the entire value of a state often does not produce the results we want. Of course our calculations have been greatly oversimplified, but it should be clear that all but the simplest sequential tasks can support at least some strong overgeneral rules.

12.5.3 Longer Sequences Increase the Bias

Now let's look at another example, in which the overgeneral matches in states R and M, and the correct rule matches only in state M. We now use the Q-values labelled d' and i' for a and i in $(c + i)/2 > a$ and obtain $76.56 > 59.05$. Notice that in this example the overgeneral acts incorrectly farther from the goal than in the first example, but its strength exceeds the threshold required of a strong overgeneral by a greater amount. The farther i and a are from the goal, the stronger the strong overgeneral will be, compared to the correct classifier. Notice also that the farther i and a are from the goal, the easier it is to produce strong overgenerals because there are more state transitions in which c can occur and gain enough strength to produce a strong overgeneral. (We can show the same thing by increasing n in equation (14).)

12.6 When Will the Value Function be Unbiased?

SB–XCS should be able to adapt to tasks with unbiased value functions, since this makes strong and fit overgenerals impossible (§8.1). Under what conditions will a value function be unbiased?

To address this, let's pose the question more carefully. In the task in figure 14, under what reward functions and values of γ will we obtain a value function which is unbiased over non-terminal states? (We do not ask that the terminal state have the same value as the other states since we defined its value to be 0.)

In terms of the reward function, the value function will be unbiased over non-terminal states only when:

$$\max_a R(s, a) = (1 - \gamma)V(s') \tag{15}$$

for all $s, s' \in \mathcal{S}$, that is, when the reward function exactly makes up for the value lost from the successor state due to discounting. Two cases where this occurs are:

1. It occurs in non-sequential tasks when the reward function is unbiased.
2. It occurs when the reward function is constant 0 over correct actions (γ can take any value).

In other words, the value function is only unbiased either in non-sequential tasks with unbiased reward functions (case 1), or in uninteresting sequential tasks with a degenerate reward function (case 2).

Note that to assert that the value function is unbiased is to assert that all states have equal value. That is, $V(s) = V(t)$ for all $s, t \in \mathcal{S}$. If this is the case, the task is effectively non-sequential, since there are no sequential decisions to make; being in any state is as good as being in any other. The only issue is what action to take, and, for a classifier system, how to generalise over states and actions.

13 What Tasks can we Solve with SB–XCS?

The circumstances under which a value function will be unbiased, and so under which SB–XCS can be expected to adapt, are limited, consisting of non-sequential tasks with relatively unbiased reward functions, and sequential tasks which are effectively non-sequential.

Furthermore, many of the unbiased non-sequential tasks for which SB–XCS is suitable are probably often better modelled as supervised learning tasks. If we can specify the correct action in each state, we have enough information to do supervised learning. Since the supervised learning paradigm provides the learner with more information (allowing it to avoid the explore/exploit dilemma), agents should be able to adapt more quickly when a task is formulated as supervised learning. Unpublished work has shown a supervised-learning-like XCS to outperform the standard XCS on the 6 multiplexer (Kovacs, 1997).

SB–XCS's prospects are particularly poor for sequential tasks. Recall that in the task in figure 14 any discounting, and any length of action sequence were sufficient to produce strong overgenerals, under our simplifying assumptions (§12.5).

This analysis suggests SB–XCS will suffer from strong and fit overgenerals in a very wide range of tasks. How much of a problem are strong and fit overgenerals? Experiments with Woods2 show that SB–XCS adapts reasonably well, but largely because much of the task is non-sequential (Kovacs, 2002). In the sequential aspects of the task, the relatively few strong overgenerals prevent it from learning an effective policy, meaning it must rely on occasional random actions to break it out of loops. In the unbiased 6 multiplexer, SB–XCS is able to adapt well but is outperformed by XCS (Kovacs, 2002), and should be outperformed by supervised learners.

This analysis would seem to leave SB–XCS, as it is, with a rather small niche. However, as we'll see in the next section, the addition of fitness sharing might make it a more useful system.

14 Extensions

This section briefly considers some extensions to the work presented earlier.

14.1 Fitness Sharing

We claim XCS avoids strong and fit overgenerals because its accuracy-based fitness penalises overgeneral rules (§9). We claim SB–XCS cannot adapt to tasks with (sufficiently) biased reward functions, because it suffers from strong and fit overgenerals. We have not, however, considered the addition of fitness sharing to SB–XCS. Fitness sharing is known to counter the propagation of overgeneral rules (Smith and Valenzuela-Rendón, 1989; Horn and Goldberg, 1998; Bull and Hurst, 2002), at least in some cases. The addition of fitness sharing to SB–XCS, and its use in other systems, may allow successful adaptation to tasks with biased reward functions, although this has yet to be

demonstrated conclusively. Clearly, this is an important direction for future work, and the analysis of rule types in this work and of representations in (Kovacs, 2002) are two possible starting points for such work.

14.2 Other Factors Contributing to Strong Overgenerals

This work has emphasised the role of the reward and value functions, and of fitness calculation, in the formation of strong and fit overgenerals. Clearly these are major factors, but there are others. Unfortunately, the analysis in this work is a gross oversimplification of more realistic learning tasks, in which it can be very difficult to determine how much of a problem strong and fit overgenerals are likely to be. Additional factors are:

the classifiers - they often apply in many states, not only two which in isolation make strong or fit overgenerals possible.

the explore/exploit policy - The strategy adopted affects how often classifiers are updated toward their different rewards.

the frequency with which given states are seen - in the non-sequential case this depends on the training scheme, and on the learner and the task itself in the sequential case.

the selection mechanisms - how high selective pressure is in reproduction and deletion.

the fitness landscape - to what extent strong and fit overgenerals compete with stronger and fitter correct rules.

As a simple example of these factors, an overgeneral might act correctly in 10 states with reward c and incorrectly in only 1 with reward i. Using the strength-based strong overgeneral inequality (theorem 6 from §6), its expected strength would be $(10c+i)/11$, and it would be a strong overgeneral if this value exceeded the strength of some accurate competitor. Similarly, the overgeneral might match in 10 states with reward i and only 1 with reward c.

Although the complexity of the issue makes a more complete analysis difficult it should be clear that the nature of the reward and value functions affect the prevalence of strong and fit overgenerals, and that they are not uncommon.

In the mainstream reinforcement learning literature strength-like values are often stored using look-up tables with an entry for each state-action pair. Such tabular systems are relatively insensitive to the form of the reward and value functions, which may account for the lack of attention this subject has received in the mainstream reinforcement learning literature. SB–XCS, however, is clearly sensitive to the form of the reward and value functions. Other strength-based LCS, even with fitness sharing, must still be influenced by the form of the reward and value functions. That is, even if fitness sharing is able to completely overcome strong and fit overgenerals, and allow strength-based LCS to adapt regardless of the form of the value function, complex value functions are still likely to be more difficult for strength-based LCS and require greater effort to learn. Fitness sharing may overcome strong overgenerals, but with some effort. This constitutes an important difference between strength-based LCS and tabular reinforcement learners. It is curious that the form of these functions has not received more attention in the LCS literature given their sensitivity to them.

14.3 Qualitative and Quantitative Approaches

We could extend the approach taken in this work by removing some of the simplifying assumptions made in §4 and dealing with the resultant additional complexity, and by including the factors in §14.2. For example, we could put aside the assumption of equiprobable states and actions, and extend the inequalities showing the requirements of the reward function for the emergence of strong overgenerals to include the frequencies with which states and actions occur. Taken far enough such extensions might allow quantitative analysis of non-trivial tasks. Unfortunately, while some extensions would be fairly simple, others would be rather more difficult.

At the same time, the most significant results from this approach may be qualitative, and some have been obtained: we have refined the concept of overgenerality and argued that strength and accuracy-based LCS have different goals (§5.3), and introduced the concepts of fit overgenerals (§5.5), and strong and fit undergenerals (§11).

We've seen that, qualitatively, strong and fit overgenerals in SB–XCS depend on biases in the reward or value function, and that they are very common. We've also seen that the newer XCS has, so far, dealt with reward function biases much better than SB–XCS (although we have not considered fitness sharing or default hierarchies). This is in keeping with the analysis in §5.3.1 which suggests that using strength as fitness results in a mismatch between the goals of the LCS and its GA. However, we have also seen that XCS is sensitive to variance in the reward and value functions and consequently it too can suffer from fit overgenerals.

In addition to these qualitative and empirical results, some interesting quantitative results have been obtained, despite our simplifications. We've seen that unbiased reward and value functions will not support strong overgenerals (sections §1 and §8.1), and we've seen the conditions under which a value function will be unbiased (§12.6).

Rather than pursue further quantitative results it would be preferable to extend the qualitative approach used here to consider the effects of default hierarchies and mechanisms to promote them, and fitness sharing. Further study of persistent strong and fit overgenerals in XCS is of interest, as are hybrid strength / accuracy-based fitness schemes, as opposed to the purely strength-based fitness of SB–XCS and purely accuracy-based fitness of XCS.

15 Conclusion

We've analysed and extended the concept of overgeneral rules under different fitness schemes. Dealing with such rules is a major issue for Michigan-style evolutionary rule-based systems in general, not just for the two classifier systems considered here. For example, use of alternative representations (e.g., fuzzy classifiers), rule discovery systems (e.g., evolution strategies) or addition of internal memory should not alter the fundamental types of rules which are possible. In all these cases, the system would still be confronted by the problems of greedy classifier creation, overgeneral, strong overgeneral, and fit overgeneral rules. Only by modifying the way in which fitness is calculated (or by restricting ourselves to benign reward functions, if they are suitable), can we influence which types of rules are possible. We've seen that strength has difficulties with biased reward functions while accuracy has difficulties with biased variance functions, so neither approach is unproblematic. It seems, however, that strength's sensitivity to biased reward functions is likely to be more limiting, particularly in sequential tasks where all non-trivial value functions are biased. Analysis of the addition of fitness sharing to SB–XCS is needed in order to gauge its effect on strong and fit overgenerals.

Although we have not described it as such, we have examined the fitness landscapes defined by the reward function, γ, task structure, rule representation and fitness scheme used. To avoid pathological landscapes we need appropriate fitness schemes.

16 Acknowledgements

Many thanks to Manfred Kerber, Riccardo Poli, Robert Smith and Stewart Wilson.

References

Bull, L. and Hurst, J. (2002). ZCS Redux. *To appear in Evolutionary Computation*.

Cliff, D. and Ross, S. (1995). Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150.

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

Horn, J. and Goldberg, D. E. (1998). Towards a Control Map for Niching. In *Foundations of Genetic Algorithms (FOGA)*, pages 287–310.

Kovacs, T. (1996). Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, University of Birmingham.

Kovacs, T. (1997). Steady State Deletion Techniques in a Classifier System. Unpublished document. School of Computer Science, University of Birmingham.

Kovacs, T. (2000). Strength or Accuracy? Fitness Calculation in Learning Classifier Systems. In Lanzi, P. L., et al., editors, *Learning Classifier Systems. From Foundations to Applications*, *LNAI* volume 1813, pages 143–160. Springer-Verlag.

Kovacs, T. (2001). Towards a theory of strong overgeneral classifiers. In Martin, W. and Spears, W. M., editors, *Foundations of Genetic Algorithms Volume 6*, pages 165–184. Morgan Kaufmann.

Kovacs, T. (2002). *A Comparison of Strength and Accuracy-Based Fitness in Learning Classifier Systems*. PhD thesis, School of Computer Science, University of Birmingham.

Lettau, M. and Uhlig, H. (1994). Rules of Thumb and Dynamic Programming. Technical report, Department of Economics, Princeton University.

Lettau, M. and Uhlig, H. (1999). Rules of thumb versus dynamic programming. *American Economic Review*, 89:148–174.

Smith, R. E. and Valenzuela-Rendón, M. (1989). A Study of Rule Set Development in a Learning Classifier System. In Schaffer, J. D., editor, *Proc. 3rd Int. Conf. on Genetic Algorithms*, pages 340–346, George Mason University. Morgan Kaufmann.

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

Venturini, G. (1994). *Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique*. PhD thesis, Université de Paris-Sud.

Wilson, S. W. (1994). ZCS: A Zeroth Level Classifier System. *Evolutionary Computation*, 2(1):1–18.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.