
Strength or Accuracy? A comparison of two approaches to fitness calculation in learning classifier systems.

Tim Kovacs

School of Computer Science
University of Birmingham
Birmingham B15 2TT United Kingdom
Email: T.Kovacs@cs.bham.ac.uk
Telephone: (44) 121 414 4773

Abstract

Wilson's XCS is a clear departure from earlier classifier systems in the way it calculates the fitness of classifiers for use in the genetic algorithm. Despite the growing body of work on XCS and the advantages claimed for it, there has been no detailed comparison of XCS and traditional strength-based systems. We distinguish different definitions of overgenerality for strength and accuracy-based fitness and analyse some implications of the use of accuracy, including an advantage in exploration. We analyse the formation of strong overgenerals, a major problem for strength-based systems, and show that they require biased reward functions. We also show that all non-trivial multi step environments have biased reward functions and thus suffer from strong overgenerals. We conclude that strength-based systems are not suitable for multi step environments or indeed many single step environments.

Keywords: accuracy-based fitness, overgeneral classifiers, strong overgeneral classifiers, XCS, complete covering maps, best action maps

1 Introduction

Reinforcement learning environments are either single step, in which case the learner's actions have no influence on which states it encounters in the future, or multi step, in which case they do. Multi step environments are more difficult as the learner has to consider the long term effects of its actions if it is to learn to behave optimally. (Multi step environments are also more likely to involve sparse and/or delayed rewards,

which tend to make learning more difficult.) In reinforcement learning we define a reward function to provide feedback on the quality of the classifier systems' actions. The reward function associates a number with each possible state/action combination in the learning environment. The goal of the learner is to obtain as much reward as possible, and the reward function is where we indicate to it what we want done. Each classifier has a strength parameter which is updated towards the rewards it receives from the environment, and is (roughly) an estimate of the reward the system will receive if it is used. When classifiers advocate conflicting actions their strengths are used in conflict resolution.

LCS have traditionally been strength-based, meaning that for the genetic algorithm the fitness of a classifier is its strength. There is a newer type of LCS called XCS which differs in that it uses accuracy-based fitness, meaning that the fitness of a classifier is based on the accuracy with which it predicts the reward the system will receive if it is used. We could say that this bases fitness on the consistency of a classifier's strength over time, as classifiers with varying strength are being successively updated towards different rewards, and so make inaccurate predictions. The creator of XCS, Stewart Wilson, gives some reasons in [1] for switching to accuracy-based fitness, but this is the only discussion in the literature and I felt a comparison of the two approaches was lacking. A better understanding of the two approaches is particularly important for a number of reasons. First, there is evidence that traditional strength-based fitness is unsuitable for multi step environments [2, 3]. Second, there is a growing body of evidence that accuracy-based fitness *is* suitable for multi step environments (e.g. [1, 9]). Third, a number of other advantages of accuracy-based fitness have been claimed, including better generalisation (and consequently smaller population sizes) [1, 4, 5]. Finally, XCS has generated considerable interest and

is becoming a major focus of LCS research (see [6] for a review). Consequently, I set out to find out in detail how the two approaches to fitness calculation compare.

To simplify the comparison, I looked at systems which were as similar as possible. I did this by starting with XCS and making the minimum changes required to convert it to a strength-based system. This involved a number of relatively minor changes to get the system to work in practice, but left the architecture of the system unchanged. This approach was necessary because there are a number of differences between XCS and other LCS, and I wanted to factor the others out and study only the difference in the fitness calculation. The resulting strength-based system is similar to Wilson’s earlier ZCS [7], but not identical to it. Unfortunately space does not permit consideration of other types of strength-based LCS, but much of the analysis should be widely applicable. Note that both the strength and accuracy-based systems considered here are Michigan style systems, both use the same Q-learning update (rather than the older Bucket Brigade algorithm), neither uses any form of tax (though they use discounting in multi step environments), classifier bids are not deducted from their strengths, and XCS does not support default hierarchies because they involve inherently inaccurate classifiers.

A consequence of accuracy-based fitness which soon becomes apparent is that *all* accurate (consistent) classifiers are maintained in the population. This includes both those which are consistently correct and those which are consistently incorrect.¹ In contrast, strength-based fitness (ideally) only maintains those classifiers which are consistently correct. As a consequence, it seems the size of the population needs to be larger if we use accuracy-based fitness, which is a disadvantage as it requires more computational power. Why would we want to maintain classifiers which have consistent predictions but which are consistently wrong?

2 Strength-based fitness

LCS researchers have been aware of two long-standing problems for many years:

Overgeneral rules For strength-based systems we define an overgeneral rule as one which matches

¹A classifier is correct when, in a given state, it advocates the best action, and incorrect when it advocates any other action. In single step environments the best action is the one which returns the most immediate reward, while in multi step environments the best action is that which leads to the most reward in the long run.

in multiple states and is incorrect in some. 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.

Greedy classifier creation Often the reward function will return different rewards for correct actions in different states, e.g. one classifier might receive a reward of 50 for acting correctly, while another might receive a reward of 100 for acting correctly. Classifiers which receive more reward from the environment have higher strength/fitness and so are more likely to be selected for reproduction than classifiers which receive less reward. If this tendency to reproduce fitter classifiers is too strong then there may be gaps in the system’s “covering map”, i.e. there may be lower reward states for which the system has no applicable classifiers. When the system encounters such states it must invent new rules on the spot, which results in poor performance (at least in the short term).

Cliff and Ross [2] showed that LCS can have serious difficulty even with simple multi step environments. They studied “classifier systems”, which at the time meant strength-based LCS as XCS had only just been invented, so their analysis applies only to strength-based systems. They attributed the LCS’s difficulties to the two problems above, and, in particular, to their interaction:

Interaction: strong overgenerals The problems of greedy classifier creation and overgeneral rules interact when an overgeneral rule acts correctly in a high reward state and incorrectly in a low reward state. The rule gains considerable strength in the state in which it acts correctly, and only misses out on a little additional strength when it acts incorrectly.² However, a rule which always acts correctly but applies only in the low reward state will have low strength. This can cause two problems. First, because action selection is done on the basis of strength the overgeneral rule will have more influence in the low reward state, despite being consistently wrong there. Second, greedy classifier creation means that the overgen-

²If classifiers pay out a bid or some form of tax then the overgeneral may lose a lot of strength when acting incorrectly. However, neither occurs in XCS because these tend to produce unstable strength values which are not good predictors of the reward to be received [1].

eral rule is more likely to be selected for reproduction, so the consistently correct rule may die out entirely. I've called this interaction effect the problem of *strong overgenerals* as Cliff and Ross did not give it a name. Strong overgenerals are disastrous for the system's performance.

Although Cliff and Ross discussed these problems with respect to the multi step case, they clearly also apply in the single step case, although not as severely as there is no possibility of disrupting sequences of actions in the single step case. Strength-based LCS have had some success with single step environments but little with multi step environments. We might attribute the difference to the greater severity of the above problems in the multi step case. However, strength-based systems can perform poorly in simple single step environments, a problem which Cliff and Ross did not discuss.

Let's extend the analysis of Cliff and Ross to see in which environments these problems can occur. First, I define a strong overgeneral as an overgeneral rule which has higher strength than some accurate rule it competes with for action selection. Two conditions must be met for strong overgenerals to emerge: i) at least two states must return different rewards (so that we can have a high reward and a low reward state), and (trivially) ii) it must be possible to act incorrectly in a lower reward state (i.e. there must be more than one action available) so that having strong overgenerals makes a difference to the learner's performance.

The reward returned for acting in a state is defined by the experimenter's reward function. Figure 1 defines two reward functions for an environment (the binary state, binary action identity function). We'll call a reward function *unbiased* if all correct actions return the same reward, and all incorrect actions return the same reward, regardless of state. (Of course the reward for correct actions needs to be higher than that for incorrect ones.) The first reward function is unbiased. The second is *biased*: more than two reward values are defined. Figure 2 shows all possible classifiers for this environment using the standard classifier representation. A & D always respond correctly, B & C always respond incorrectly, and E & F are overgeneral, responding correctly in one state and incorrectly in the other. If we assume that states and actions are chosen equiprobably we obtain the expected strength values shown for the classifiers using the two reward functions. Using the biased reward function E is a strong overgeneral: despite being wrong half the time it has higher strength than D (which is always correct).³

³We could have simplified the example somewhat by defining only action 0 for state 0. Action 1 has no effect on

Having a biased reward function *can* lead to the problems of greedy classifier creation and strong overgenerals which Cliff and Ross discussed. However, it is difficult to determine just when these problems will occur, and how serious they will be because there are many factors involved. An important factor is the degree of bias in the reward function. Let's begin a simplified analysis by labelling the rewards from the overgeneral rule's perspective. Let c be the reward when the overgeneral acts correctly (state 0 action 0 in the biased reward function), i the reward when acting incorrectly (state 1 action 0), and a the reward for the accurate rule which applies only in the lower reward state (state 1 action 1).

Now let's see what reward functions will cause E to be a strong overgeneral. Assuming E is updated towards c and i such that its strength is the average of the two (which is an oversimplification), it will be a strong overgeneral if $(c + i) / 2 > a$. Solving for c yields: $c > 2a - i$. If we use the values of 0 and 200 for i and a as in the biased reward function then $c > 400$ will result in E being a strong overgeneral.

Clearly the rewards defined by the reward function play a major role in determining whether strong overgenerals are possible. Unfortunately, the above analysis is a gross oversimplification of more realistic learning problems, in which it can be very difficult to determine how much of a problem strong overgenerals are likely to be.

Thus far I have not dealt with cases in which we have more than two actions. In such cases we may be tempted to use more than two reward levels (see section 3), but this gives us a biased reward function.

2.1 Multi step environments

In single step environments a reinforcement learner approximates the reward function defined by the experimenter, which is in principle enough to maximise the reward it receives. In multi step environments, however, consideration of the reward function alone is not sufficient, as it defines immediate reward (the reward on a given time step) only. In multi step environments the learner's actions influence the state of the environment 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 received. Q-learners do so by learning a *Q-function* (also called a state/action function) which maps state/action pairs to an estimate of their long term value (called their Q-

the development of strong overgenerals in this example.

State	Action	Reward
0	0	1000
0	1	0
1	0	0
1	1	1000

State	Action	Reward
0	0	1000
0	1	0
1	0	0
1	1	200

Figure 1: Unbiased reward function (left) and biased reward function (right).

Classifier	Condition	Action	Strength
A	0	0	1000
B	0	1	0
C	1	0	0
D	1	1	1000
E	#	0	500
F	#	1	500

Classifier	Condition	Action	Strength
A	0	0	1000
B	0	1	0
C	1	0	0
D	1	1	200
E	#	0	500
F	#	1	100

Figure 2: Classifiers using the unbiased and biased reward functions (left and right respectively).

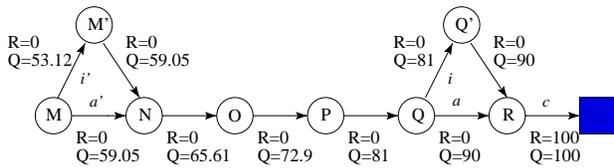


Figure 3: A simple multi step environment showing immediate rewards R and Q -values Q for state transitions using $\gamma = 0.9$.

value). (Q stands for quality of the state/action pair.) The Q -value of a state/action pair is updated towards the immediate reward it receives plus some fraction of the Q -value of the state which follows it. (We say the value of the following state is discounted and passed back to its predecessor.) In this way a state/action pair which receives no immediate reward will have a Q -value greater than 0 if it leads to a state with an immediate reward greater than 0. The value of a state is the value of the best action for that state, i.e. the state/action pair with the highest Q -value. Discounting is illustrated in a simple environment in figure 3 which shows the immediate reward R and the Q -value Q for each state transition assuming a discount rate γ of 0.9. Knowledge of the true Q -function for an environment is sufficient to act optimally by simply taking the action with the highest associated Q -value.

In an LCS using Q -learning the approximated Q -function is represented by classifiers whose strengths are Q -values. Discounting is necessary in order to find shorter paths through sequences of states, but we shall see that the use of discounting in Q -learning introduces special problems for strength-based LCS.⁴

⁴Classifier systems have traditionally used the Bucket Brigade algorithm rather than Q -learning, but it has the

The discount rate γ controls how much consideration the system gives to future rewards in making decisions. At $\gamma = 1.0$ no discounting occurs, and the system will learn the path 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. If we used $\gamma = 1.0$ in figure 3 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 is often undesirable, as it would lead the system to choose £5 today rather than £1000 tomorrow. In figure 3, $\gamma = 0.0$ 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. However, this produces one of the two criteria for the production of strong overgenerals: a biased Q -function.⁵ Notice that even though there are only two rewards given in figure 3, there are many Q -values.

A further factor is often involved in generating strong overgenerals in multi step environments. Suboptimal paths are routes through the state space which are longer than necessary. (It is easy to generate them in environments with circularities, but circularities are not necessary.) The combination of suboptimal paths and discounting produces states with highly biased Q -values. Let's look at an example. Imagine the situation where an overgeneral matches in state R and

same need for discounting.

⁵The other criterion, that it be possible to act incorrectly, occurs in all non-trivial environments, specifically, in all environments in which we have a choice of alternative actions in at least one state.

advocates a transition to the terminal state, and also matches in Q and advocates an incorrect (suboptimal) transition to Q'. Additionally, a different, accurate, rule matches only in state Q and advocates the correct transition to state R. Assigning the Q-values from the transitions so labelled to c , i and a in the inequality $(c+i)/2 > a$ we saw earlier we obtain $(100+81)/2 > 90$ or $90.5 > 90$. In other words, the overgeneral is a strong overgeneral as it has strength 90.5 which exceeds the accurate rule's strength of 90. This indicates that strong overgenerals can be obtained even with very short chains.

If the reward received at some state s is $R(s)$, discounting results in $R(s)\gamma$ being passed to the state preceding s . In general, a state n steps ahead of s receives $R(s)\gamma^n$. If we rewrite a as $c\gamma^n$ and i as $c\gamma^m$ where n and m are integer distances of a and i from c , we have $(c + c\gamma^m)/2 > c\gamma^n$. This expression is true for any $c > 0$ as long as $0 < \gamma < 1$ and $n \geq 1$. In other words, according to our approximate expression, *any* discounting will produce a Q-function capable of supporting strong overgenerals in this environment. Of course our calculations have been greatly oversimplified, but it should be clear that all but the simplest multi step environments can support at least some strong overgenerals.

Now let's look at another example, in which the overgeneral matches in states R and M, and the accurate rule matches only in state M. We now use the Q-values labelled a' 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 degree. The farther i and a are from the goal, the stronger the strong overgeneral will be, compared to the accurate 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.

3 Why bias the reward function?

The reward function is the means by which the experimenter defines the goals of the learner (see [8]). The reward function is part of the definition of the problem; choosing different reward functions results in different learning problems. The trick is to choose a reward function which results in the desired behaviour. Given that the experimenter defines the reward function, potential problems with biased reward functions can be avoided by using unbiased ones (or only slightly biased

ones). One reason to bias the reward function is to get the system to allocate more resources (classifiers) to more important parts of the environment. For example, we might consider the detection of an impending meltdown to be more important than optimising the output of a reactor.

In single step environments we can normally achieve the same effect by varying the frequency with which individual states are encountered. (Although we might not be able to do this if we're learning on-line and don't have enough memory to store exemplars for later playback.) Unfortunately, this is not an option in multi step environments as the agent and environment determine which states are encountered, not the experimenter. Alternatively, we could split the classifiers into multiple subpopulations, each with a different size limit, and assign each subpopulation to a subpart of the environment. More important parts of the environment would get bigger subpopulations.

Another reason to bias the reward function in single step environments is to indicate the relative value of different actions when more than two are available. Suppose we can take many actions. We could give a high reward (e.g. 100) for the best action in each state and a low reward (e.g. 0) for all others. This would be an unbiased reward function. However, we might not want to treat all sub-optimal actions equally. Some might be much better than others.

In multi step environments we don't need to bias the reward function to indicate the relative value of different actions (as they relate to the same goal) because discounting does this for us by producing a biased Q-function. This bias is towards classifiers which apply in states which are closer to the source of the reward. It is not clear to me whether this bias is desirable or not. However, there is an additional need to purposely bias the reward function in multi step environments because the learner's actions affect which states it will see in the future. If we define multiple goals we need to tell the learner about their relative importance, so it knows which goals to pursue if it must choose between them.

4 Accuracy-based fitness

The only fully accuracy-based LCS is Wilson's XCS. (See [1] for a review of the use of accuracy in primarily strength-based systems.) Although the difference in fitness calculation may seem minor, it has profound implications for the system. Strength-based systems attempt to find rules which advocate the best action for each state. Ideally they would maintain only a sin-

gle rule (advocating the best action) for each state. We call this a *best action map*. Since the action selection mechanism can only choose between the advocated actions, the maintenance of a best action map means the genetic algorithm has a hand in action selection. In contrast, the idea in XCS is to find a population of rules such that each action in each state is advocated. We call this a *complete map*. It is then up to the action selection mechanism to decide which action to take. Thus, in XCS, the genetic algorithm is dissociated from action selection. Its role is only to search for useful generalisations over states. I use different definitions for overgeneral rules to match the different objectives of the two types of system.

For accuracy-based fitness I define an overgeneral rule as one which receives different rewards in different states. As noted earlier, it is possible for the reward function to be biased in such a way that a rule receives the same reward when acting correctly in one state and incorrectly in another. Such a rule would have a consistent prediction and thus be accurate, fit and *not* overgeneral in an accuracy-based system. However, it would be overgeneral in a strength-based system as it advocates an incorrect action. In any case, this seems unlikely to occur in practice.

A much more significant difference is that a rule can successfully advocate the same correct action in states with different Q-values only using strength’s definition of overgenerality. For example, a rule which matches in states O and P in figure 3 will be updated towards two Q-values: 72.9 and 81. Its strength will be some sort of average of these values. With fitness based on strength, this rule will have a reasonable fitness. But with fitness based on accuracy, this rule will have low fitness, as its prediction will oscillate between the two Q-values and be a good estimate of neither.

Accuracy-based fitness can only generalise over state/action pairs with very similar Q-values.⁶ Strength-based fitness does not have this limitation. Strength-based classifiers are free to match any and all states for which their action is optimal. Unfortunately, they are also free to match states for which their action is suboptimal, and, in the standard strength-based system, there is nothing preventing them from doing so.

According to Wilson’s Generalization Hypothesis [1] there is a tendency in accuracy-based XCS for the more general of two equally accurate rules to reproduce more. This results in a tendency to evolve *maximally general rules*; rules which cannot be made any more general without becoming inaccurate. These ac-

⁶This is a limitation of dynamic programming based systems rather than of XCS or LCS per se.

curate, general rules form a compact representation of the learned solution to the problem environment. Consequently, XCS maintains populations which are smaller than those of standard strength-based systems. In fact, with minor extensions XCS is able to consistently evolve *optimal solutions* for boolean (multiplexer and parity) functions (see [4]). These are solutions which are complete (cover all parts of the input/action space), non-overlapping (no part of the space is described more than once) and minimal (the minimal number of non-overlapping rules is used).

In addition to a tendency towards best action maps, strength has two other advantages over accuracy in maintaining small population sizes. First, accuracy cannot generalise over correct actions which result in different rewards. Second, accuracy cannot support normal default hierarchies because default hierarchies involve classifiers which are inherently inaccurate.⁷ It is not clear at present how the two compare in practice in terms of population size.

An important property of accuracy-based systems is that they are largely insensitive to bias in the reward function. This is because fitness is based on the accuracy (consistency) of the reward prediction, rather than the magnitude of the prediction. So a classifier which accurately predicts a reward of 0 will be as fit as one which equally accurately predicts a reward of 100 – it does not matter much what values we use in the reward function (as long as correct rules have higher reward than incorrect rules). So accuracy-based fitness avoids the problem of greedy classifier creation. It also largely avoids problems with overgenerals because they are updated towards different rewards and so have low accuracy and low fitness. In particular, it should be difficult for strong overgenerals to emerge using accuracy because overgenerals tend to have low fitness while all accurate classifiers have high fitness.

However, insensitivity to bias in the reward function also means we are unable to bias the allocation of classifiers towards more important parts of the environment by biasing the reward function. With accuracy-based fitness there is no point in using a biased reward function.⁸ However, we can still bias the allocation of rules in other ways, as discussed in section 3. Another option, which is applicable in any situation, is simply to define a weighting function for each state/action

⁷However, it should be possible for accuracy to use classifiers with multiple conditions which form their own internal default hierarchies.

⁸In fact it is a little easier for the system to generalise if the reward function is unbiased because, with a biased reward function correct rules can receive different rewards and thus cannot be generalised over.

pair, and to modify the system to take it into account when allocating rules. This reintroduces the possibility of problems with greedy classifier creation.

In multi step environments discounting always produces a biased Q-function, but this does not matter as accuracy-based fitness is largely insensitive to it and appears to work well in multi step environments (as demonstrated in a number of environments in e.g. [1, 9]). I said earlier (section 3) that we might need to bias the reward function in multi step environments in order to give different goals relative importance. Although accuracy-based fitness means the system is insensitive to bias when it comes to reproducing classifiers with the genetic algorithm, action selection is still done on the basis of strength, not accuracy-based fitness. Thus, an accuracy-based system can distinguish between high and low priority goals (or find shortest paths to goals).

5 Complete maps and best action maps

Now let's return to the question of why we would want to maintain classifiers which are consistently wrong. Accuracy-based fitness maintains all accurate classifiers (both consistently right and consistently wrong). For any state, an accuracy-based system will tend towards having at least one classifier advocating each possible action (a complete map). Strength-based systems, in contrast, tend towards maintaining only a single classifier for each state, namely the one which advocates the action which results in the highest reward (a best action map). Thus it seems that accuracy-based systems should require a larger population of classifiers, although whether this is the case is not clear. (The worst case is that the complete map will be n times larger than the best action map, where n is the number of actions available. In practice, however, the difference will be less as strength only tends towards a best action map.)

However, maintaining consistently wrong classifiers in the population does not mean we have to take the wrong action each time that classifier is applicable. In fact, quite the opposite is true. If a classifier is consistently wrong, this suggests we should *not* take the action it advocates. If we did not keep this consistently wrong classifier in the population, how would we know it was a bad idea to take its action? If we delete it, we have no record of the utility, or lack thereof, of taking that action in that state, and we might be tempted to try the bad action again and again in the future. We need to keep bad classifiers in order to keep track

of our failed guesses; in other words to manage our explore/exploit strategy.

Accuracy-based fitness tends towards forming and maintaining complete covering maps because it is insensitive to the magnitude of reward received by a classifier. Strength-based fitness, in contrast, tends to maintain classifiers for only a subset of the actions for each state. Can we get a strength-based system to maintain a complete map? We could use a reward function which, for example, gave rewards of 90 for incorrect actions, and 100 for correct actions. If selective pressure is not too strong the system should be able to maintain classifiers for both correct and incorrect actions (i.e. a complete map). A problem is that over-general rules would always have more strength than accurate incorrect rules.

A complete map has the disadvantage of requiring more classifiers but the advantage of helping with the explore/exploit problem. Whether the advantage outweighs the disadvantage depends on the situation. If larger population sizes are costly (e.g. as on a serial computer) then a complete map is costly (although the power required is likely to increase by only a small polynomial factor). On the other hand, the more difficult the exploration problem, the more advantageous a complete map will be. Two cases in which exploration is more difficult are i) when the environment changes over time, and ii) in multi step environments.

Hartley [10] trained two LCSs, XCS (which has a complete map) and NEWBOOLE (which has a best action map), on a binary categorisation task, then abruptly switched the category each stimulus belonged to. XCS quickly recovered from these changes by simply adjusting the strengths of the rules involved: consistently correct rules suddenly became consistently incorrect and vice versa. NEWBOOLE, in contrast, found that its rules suddenly all had low strength, and had to engage the genetic algorithm to generate new ones as it does not maintain low strength rules. A complete map should also be useful for adapting to less systematic and more gradual changes in the environment, although this has not been studied.

In multi step environments globally optimal behaviour often requires that the learner take locally suboptimal actions (i.e. actions which do not return the highest possible immediate reward). E.g. the learner must take action B, knowing that action A results in greater reward, because only action B will lead it to state Q where it can obtain even greater reward. The best way to ensure that such sequences of actions can be learned is to use a complete covering map. Note that a tabular Q-learning system employs a complete map.

Further study is needed to confirm and quantify the advantages of complete covering maps for exploration control, and to determine how much of an increase in population size is required by a complete map.

6 The big picture

That an estimate of accuracy is needed should not be surprising because LCS are searching a space of generalisations and accuracy is a measure of the utility of generalisation. Function optimisation genetic algorithms are rather different. Typically they attempt to find parameter settings which correspond to the extremum of the fitness function (or, in LCS terminology, the reward function). They include no concept of environmental state (and so lack the condition part of a classifier), and only manipulate a set of parameters (corresponding to the action part of a classifier). Consequently they have no notion of generalising across states and no need for a measure of the accuracy with which this is done. Their measure of fitness is just the strength value in a strength-based classifier system (with the important difference that chromosomes in a genetic algorithm are normally only evaluated once, whereas a classifier's strength is updated many times).

The use of strength as a fitness measure in LCS is apparently due to confusion over the role of the genetic algorithm. In function optimisation, the genetic algorithm is a function optimiser seeking only the extremum of the fitness function. Within an LCS, however, it is a function approximator, learning to duplicate the entire reward function, hopefully doing so more efficiently by generalising over states. It was easy to make the mistake of retaining the same fitness measure when moving from one to the other.

7 Summary of comparison

I've tried to show that strength-based fitness is not suitable for non-trivial multi step environments. The only potential advantage of strength we've seen is that it can, in principle, maintain smaller populations. However, it is not clear how the approaches really compare in this respect. Further, it seems that complete maps are important for controlling exploration, so strength may be well suited only to single step environments in which exploration is relatively easy. Accuracy-based fitness seems more suitable for more difficult single step environments, and appears to be necessary for non-trivial multi step environments.

I have evaluated the accuracy-based fitness of XCS while ignoring its other valuable features.

8 Acknowledgements

I am grateful to Stewart Wilson, Manfred Kerber and two anonymous reviewers for their comments.

References

- [1] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. <http://prediction-dynamics.com/>
- [2] Dave Cliff and Susi Ross. Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150, 1995. <ftp://ftp.cogs.susx.ac.uk/pub/reports/csrp/csrp347.ps.Z>
- [3] Tim Kovacs. Weeding populations of classifiers. In preparation. <http://www.cs.bham.ac.uk/~tyk>
- [4] Tim Kovacs. XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, 1997. <ftp://ftp.cs.bham.ac.uk/pub/authors/T.Kovacs/>
- [5] Stewart W. Wilson. Generalization in the XCS classifier system. In J. Koza et al., editor, *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann, 1998.
- [6] Stewart W. Wilson. State of XCS classifier system research. Technical Report 99.1.1, Prediction Dynamics, Concord MA, 1999.
- [7] Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1), 1994.
- [8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. <http://www-anw.cs.umass.edu/~rich/book/the-book.html>
- [9] Pier Luca Lanzi and Marco Colombetti. An Extension of XCS to Stochastic Environments. To appear in W. Banzhaf et al. eds, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999.
- [10] Adrian Hartley. Accuracy-based fitness allows similar performance to humans in static and dynamic classification environments. To appear in W. Banzhaf et al. eds, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999.