Generalized State Values in an Anticipatory Learning Classifier System

Martin V. Butz[1,2] and David E. Goldberg[1]

[1] Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign, IL, USA
{butz,deg}@illigal.ge.uiuc.edu
[2] Department of Cognitive Psychology
University of Würzburg, Germany
butz@psychologie.uni-wuerzburg.de

Abstract. This paper introduces generalized state values to the anticipatory learning classifier system ACS2. Previous studies with ACS2 showed that the system reliably evolves an accurate and generalized predictive model in typical Markov decision process (MDP) environments. The predictive model approximates the state transition function of the MDP in a compact, generalized form. However, it was also shown that the evolving generalized predictive model might be over-general for an accurate representation of reinforcement values. Thus, a function approximation module is added that approximates state values. In combination, actual action choice depends on state values predicted by the means of the predictive model yielding anticipatory behavior. It is shown that the function approximation module accurately generalizes the state value function in the investigated MDP. We also suggest the implementation of task dependent anticipatory attentional mechanisms exploiting the representation of the generalized state-value function. Moreover, improvement of the approach by the means of further anticipatory interaction between predictive model learner and state value learner is suggested.

1 Introduction

Anticipatory learning classifier systems (ALCSs) are one form of anticipatory learning systems that learn a generalized predictive model of an environment online. Predictive knowledge in ALCSs is usually stored in rules, the so called classifiers. This paper introduces one approach for the simulation of anticipatory behavior in an ALCS using generalized state values.

Previous studies have learned state values with other ALCSs (Gérard & Sigaud, 2001b; Gérard, Meyer, & Sigaud, 2002) exhibiting optimal behavior. In these approaches all encountered states are stored explicitly in a state list. Value iteration is applied over the stored states learning an optimal policy represented in the state values. None of the approaches tried to generalize the state list, though.

ACS2 is an ALCS that recently showed to reliably evolve a complete, accurate, and compact predictive model in many typical MDP environments. However, it was also shown that ACS2 suffers from a *model aliasing* problem. In model aliasing, the learned predictive model is accurate in terms of the prediction of resulting states but inaccurate in terms of predicting reinforcement. In this paper we introduce a generalizing state value learner to ACS2 (Butz, 2002b). The combination of ACS2 with a generalizing state value learner is termed XACS. XACS shows optimal behavior in a blocks world problem in which ACS2 was not able to learn an optimal behavior due to model aliasing. Besides the achieved optimal behavior, we also show that the generalization mechanism in XACS, currently implemented by the classifier system XCS (Wilson, 1995), works efficiently. We confirm that the generalization properly specifies task relevant perceptual attributes. Moreover, generalization enables learning of problems with larger state spaces. The implementation of other anticipatory mechanisms in XACS, such as task dependent attentional mechanisms or further interaction of the learning components, are suggested.

The paper starts with a short introduction to the anticipatory learning system ACS2. Next, the problem of model-aliasing is explained and the blocks world problem is introduced along those lines. Section 4 introduces the function approximation module, implemented by the accuracy based classifier system XCS, that approximates the state values for an optimal anticipatory behavioral policy. The derived system XACS is tested in the blocks world problem confirming optimal performance and proper generalization. Section 6 outlines how further anticipatory mechanisms could be included in XACS. Finally, summary and conclusions are provided.

2 ACS2 — A Predictive Model Learner

ALCSs learn a predictive model of an encountered environment specifying the consequences of each possible action in each possible situation. Predictions are usually represented explicitly in each rule or classifier. That is, each rule consists of a condition-action-effect triple that specifies which perceptual effects are expected after executing the specified action given the specified conditions.

A first learning classifier system, termed ACS, with an additional anticipatory part in each classifier was published in (Stolzmann, 1998; Stolzmann, 2000). The ALCS investigated in this paper, termed ACS2, is an enhancement of Stolzmann's ACS including a genetic generalization mechanism. A complete description of the system including an algorithmic description as well as an experimental performance analysis can be found in Butz (2002a). This section gives a basic introduction to the system. The intention is to give the reader the idea of what the system is actually learning and how it basically works. For more details, the interested reader is referred to the cited literature.

2.1 Agent architecture

ACS2 interacts with an environment or problem perceiving situations and reinforcement and executing actions. In reinforcement learning terms, ACS2 learns the state transition function of an MDP.

More formally, ACS2 perceives situations $\sigma \in \mathcal{I} = \{\iota_1, \iota_2, ..., \iota_m\}^L$ where m denotes the number of possible values of each environmental attribute and L the number of attributes. ACS2 executes actions $\alpha \in \mathcal{A} = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ where n specifies the number of different possible actions in the environment and $\alpha_1, ..., \alpha_n$ denote the different possible actions. The environment is characterized by its state transition function f that transfers current states and actions into consequent states: $f : \mathcal{I} \times \mathcal{A} \to \mathcal{I}$. ACS2 learns an approximation of the state transition function. We assume the Markov property of a problem, that is, the consequences of any action in any situation only depend on the current perceived situation and not on the history. Moreover, we restrict ourselves to deterministic environments. Thus, any situation-action combination $(\sigma - \alpha)$ deterministically leads to a new situation σ'.

Moreover, reinforcement is provided in the MDP. That is, the environment provides scalar feedback ρ for specific situation, action, next situation combinations. The feedback ρ is determined by a reinforcement function $\nabla : \mathcal{I} \times \mathcal{A} \times \mathcal{I} \to \Re$. Thus, the environment represents an MDP with states $\mathcal{I}$, actions $\mathcal{A}$, a reward function ∇, and a state transition function f.

2.2 Knowledge Representation

Knowledge in ACS2 is represented by a *population* $[P]$ of condition-action-effect rules, i.e. the *classifiers*. A classifier in ACS2 predicts a complete resulting state. Each classifier further carries the following additional attributes. The *quality* q measures the average accuracy of the effect prediction of the classifier. The *reward prediction* r estimates the average discounted reward encountered. The *immediate reward prediction* ir estimates direct reward. Moreover, a classifier carries a *mark* M, a *GA time stamp* t_{ga}, an anticipatory learning process (ALP) *time stamp* t_{alp}, an *application average estimate* aav, an *experience* counter exp, and a *numerosity* counter num.

Condition and effect part consist of the values perceived from the environment and '#'-symbols (i.e. $C, E \in \{\iota_1, ..., \iota_m, \#\}^L$). A #-symbol in the condition, called *don't-care symbol*, denotes that the classifier matches any value in this attribute. A '#'-symbol in the effect part, called *pass-through symbol*, specifies that the classifier anticipates that the value of this attribute will not change after the execution of the specified action. Non pass-through symbols in E anticipate the change of the particular attribute to the specified value in contrast to ACS in which a non pass-through symbol did not require a change in value. Action parts specify any action possible in the environment ($A \in \mathcal{A}$). The measures q, r, and ir are scalar values where $q \in [0, 1]$ and $r, ir \in \Re$. For further information on the other parameters, the interested reader is referred to Butz (2002a). Classifier attributes are modified by the means of two model learning mechanisms and a reinforcement learning mechanism explained below.

2.3 Learning in ACS2

While interacting with an environment, the population of ACS2 increasingly approximates the state transition function of the MDP. Usually, the agent starts without any prior knowledge except for the knowledge implicitly included in the coding structure and the provided actions. Initially, classifiers are mainly generated by a covering mechanism. Later, an anticipatory learning process (ALP) generates specialized classifiers while a genetic generalization process produces generalized offspring. RL techniques are applied for the evolution of an optimal behavioral policy represented directly in the predictive model.

Figure 1 illustrates the interaction of ACS2 with its environment and its learning application in further detail. After the perception of the current situation $\sigma(t)$, ACS2 forms a match set $[M]$ comprising all classifiers in the population $[P]$ whose conditions are satisfied in $\sigma(t)$. Next, ACS2 chooses an action $\alpha(t)$ according to some action selection strategy and an action set $[A]$ is generated that consists of all classifiers in $[M]$ that advocate $\alpha(t)$. After the execution of $\alpha(t)$, classifier parameters are updated by ALP and reinforcement learning techniques and new classifiers might be generated as well as old classifiers might be deleted by ALP and genetic generalization.

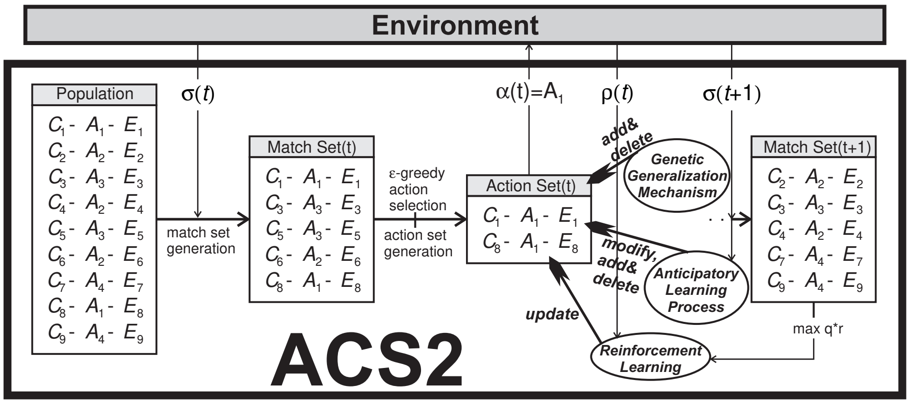

Fig. 1. During one agent/environment interaction, ACS2 forms a match set representing the knowledge about the current perceptions. Next, it generates an action set representing the knowledge about the consequences of the chosen action in the given situation. Classifier parameters are updated by RL and ALP. The model evolves by the means of ALP and genetic generalization.

The ALP is derived from the cognitive learning theory of anticipatory behavioral control (Hoffmann, 1993). The process evaluates and specializes the predictive model of ACS2. Parameters are updated evaluating the correctness of the effect parts of classifiers in the current action set $[A]$ with respect to the subsequent perception $\sigma(t+1)$. Moreover, specialized offspring is generated where necessary and inaccurate classifiers are deleted. Parameter updates are done using learning rate β. For example, if a classifier predicted the subsequent perception correctly, its quality is increased by $q \leftarrow q + \beta(1-q)$. If there was no representative present for the chosen action, a *covering classifier* is generated that is specialized in all attributes of condition and effect part that changed from $\sigma(t)$ to $\sigma(t+1)$.

Despite the informed specialization process by the ALP, over-specializations can occur sometimes as studied in (Butz, 2002a). Since over-specialization can be the result of several distinct circumstances, a genetic generalization (GG) mechanism is applied. GG selects classifiers in action sets $[A]$ proportionally to the fitness. Selected classifiers are reproduced and the conditions are generalized (by a generalizing mutation operator) and crossed over. Finally, classifiers are deleted in $[A]$ if the action set size exceeds a certain threshold θ_{as}.

The interaction of ALP and GG results in the generation of a complete, accurate, and maximally general environmental model. While ALP continuously specializes over-general classifiers, GG continuously generates generalized offspring. Thus, there is a continuous generalization pressure that alleviates possible over-specializations. Moreover, possible over-generalizations are alleviated by the specialization pressure where necessary. Further balancing is achieved by a subsumption technique in which over-specialized offspring is subsumed by existing accurate more general classifiers.

The applied RL technique is an approximation of Q-learning (Watkins, 1989). The reward prediction r of a classifier is updated according to the following equation:

$$r \leftarrow r + \beta(\rho(t) + \gamma \max_{cl \in [M](t+1) \wedge cl.E \neq \{\#\}^L} (cl.q \cdot cl.r) - r) \tag{1}$$

Parameter $\beta \in [0,1]$ denotes the learning rate biasing the parameters more or less toward recently encountered reward values. $\gamma \in [0,1)$ denotes the discount factor biasing the reward values more or less toward future reinforcement values. The reward prediction value r consequently estimates the average resulting discounted reward after executing action A in all possible situations σ in which the classifier is applicable.

Action selection can now be biased more or less strong on the reward prediction values r. Usually, ACS2 applies a simple ϵ-greedy action selection strategy (Sutton & Barto, 1998) in which an action is chosen at random with a probability ϵ and otherwise the best action is chosen. The action of the classifier with the highest qr value in a match set $[M]$ is usually considered as the best action in ACS2. Recently, an additional action selection bias was applied speeding up model learning (Butz, 2002b). Instead of choosing an action at random, with a probability p_b an action is chosen that promises the highest knowledge increase (represented either by the highest average application delay or the lowest average quality).

3 Model Aliasing

While the evolving behavioral policy represented by the reward prediction values of classifiers previously showed to solve different maze problems, it was also shown that a problem, termed *model aliasing*, can occur. The problem is that the classifiers in the the evolving predictive model might be over-general to specify accurate reinforcement values. That is, although a classifier might accurately specify the perceptual effects in all situations its conditions are satisfied, its reward prediction value might be inaccurate. Similar problems have been reported in the reinforcement learning literature when generalizing the state value function (Boyan & Moore, 1995; Kaelbling, Littman, & Moore, 1996).

3.1 A Simple Example

An example with respect to model aliasing might clarify the problem. Imagine a game such as four-wins in which ACS2 learns the moves of the game. Actions would be to place a coin in either of the (usually) seven slots. After some time, ACS2 would have a proper representation of the effects of all possible actions (representing the moves of the other player in some way). To make accurate action-effect predictions the condition of a classifier must specify the current number of coins in the slot the specified drop-coin action in the classifier aims for. The conditions, however, would be over-general to specify an accurate reinforcement value since the accurate classifiers are applicable nearly independent of the game situation. Thus, the reward prediction values would average over all possible actual game situations and would consequently be meaningless. The basic intuition behind this is that a representation of the rules of a game are usually insufficient for the representation of a good strategy for playing the game.

3.2 Blocks World Problem

We investigate the behavior of ACS2 in a blocks world scenario introduced in Butz (2002a) in which a model aliasing problem applies similar to the four-wins game. In the blocks world, b blocks are distributed over a certain number of stacks s. ACS2 is able to manipulate the stacks by the means of a gripper that can either grip or release a block on a certain stack. ACS2 perceives the current block distribution coding each stack with b attributes. One additional attribute indicates if the gripper is currently holding a block. Thus, the perceivable situations are a subset of $\mathcal{I} \subset \{*, b\}^{bs+1}$. Since there are $\binom{b+s-1}{s-1}$ possibilities to distribute b blocks over s stacks and in a current situation ACS2 might currently hold a block in its gripper or not, there are $\binom{b+s-1}{s-1} + \binom{b+s-2}{s-1}$ possible environmental situations.

Given the task to have exactly y blocks on the first stack, the model is usually over-general to represent an optimal behavioral policy. Figure 2 visualizes the problem in a simple $b = 4, s = 3, y = 4$ blocks world. To predict the effect of releasing a block on the middle stack accurately, the condition of a classifier needs to specify only that there is no block present on the middle stack and a block is currently gripped. This classifier matches in all four situations shown at the left of Figure 2. Thus, its reward prediction value will approximate an average value over all those possible situations estimating that $(8+6+4+2)/4 = 5$ steps are necessary to reach the goal. Releasing a block on the left stack in the left-most situation requires exactly 6 further steps to reach the goal. Thus, dropping a block on the middle stack would be preferred exhibiting the model-aliasing problem. Additionally, the model-aliasing problem is propagated by the means of the adapted $Q-learning$ mechanism.

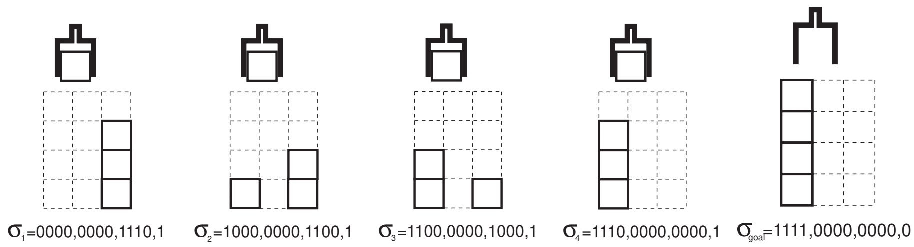

Fig. 2. Model Aliasing in ACS2: To accurately predict the effects of a releasing action on a particular stack, classifiers in ACS2 evolve whose conditions only specify the state of the addressed stack. Thus, given the goal shown in the right-most scenario, the classifier for releasing a block on the first stack will have a lower reward prediction value than the classifier for releasing a block on the second stack in the left-most scenario since the latter classifier is applicable in all four scenarios on the left.

The next section introduces a function approximation module that approximates the optimal state value function making the policy representation independent of the predictive model representation.

4 Generalized State Values in ACS2

As the last section revealed, the RL approach in ACS2 is somewhat limited. The approach is only capable of solving Markov decision processes in which the learned predictive model is specific enough to represent accurate reinforcement values. This section is dedicated to remedy this drawback introducing a generalizing state value learner implemented by a modified version of the learning classifier system XCS (Wilson, 1995).

Previously, Gérard and Sigaud (2001b) published an approach to the same problem using a list of all encountered states to learn a proper state value approximation. Our approach takes this idea one step further by using a generalized representation of states. We first formalize the actual state values that ACS2 learns to behave optimally. Next, we introduce XCS that learns a generalized representation of these state values.

4.1 Optimal Policy with State Values and Model

In the MDP framework formulated above, reinforcement is provided in certain situation, action, resulting situation combinations. Since ACS2 learns a predictive model, i.e. the state transition function f of the MDP, dynamic programming (DP) techniques can be applied using ACS2 to approximate the following Bellman equation representing the state value function for an optimal behavioral policy.

$$V^*(\sigma) = R(\sigma) + \gamma \max_{\alpha} V^*(f(\sigma, \alpha)) \tag{2}$$

where

$$R(\sigma) = \frac{\sum_{\{(\sigma^{-1},\alpha^{-1})|f(\sigma^{-1},\alpha^{-1})=\sigma\}} \nabla(\sigma^{-1}, \alpha^{-1}, \sigma)}{|\{(\sigma^{-1}, \alpha^{-1})|f(\sigma^{-1}, \alpha^{-1}) = \sigma\}|}$$

Hereby, $R(\sigma)$ denotes the average immediate reinforcement encountered when entering state σ. This notation differs from the usual state value definition. It enables us to disregard a separate representation of immediate reward.

The optimal policy can be defined based on the state transition function f and the optimal state values V^*.

$$\pi^*(\sigma) = \arg\max_{\alpha} V^*(f(\sigma, \alpha)) \tag{3}$$

The optimal strategy π^* evolves by iteratively learning the optimal state value function in the following way.

$$V(\sigma) = V(\sigma) + \beta(\rho(t) + \gamma \max_{\alpha} V(f(\sigma, \alpha)) - V(\sigma)) \tag{4}$$

Parameter β denotes the learning rate. For a proper convergence, it must be assured that ρ is sampled uniformly. Since in the problems investigated below $\nabla(\sigma^{-1}, \alpha^{-1}, \sigma)$ is equal for all combinations of σ^{-1} and α^{-1} this is not a problem in this experiments.

As explained above, ACS2 approximates the state transition function f of the environment. Once ACS2 represents the complete state transition function f accurately, the updates can be determined using ACS2's model instead of the actual state transition function f. Since model and policy should be learned simultaneously, the state transition function is approximated by ACS2's best guess in a given situation-action combination. As the best guess, denoted by bg, we use the highest quality classifier of ACS2 of all classifiers in ACS2 that match in the given situation and specify the corresponding action.

$$\mathrm{bg}(\sigma, \alpha) = \arg \max_{\{cl|cl \in [P] \wedge cl.C \text{ matches } \sigma \wedge cl.A = \alpha\}} cl.q \tag{5}$$

Value updates in XCS are now applied in all encountered situations using the prediction of classifier cl, $p(\sigma, \alpha, cl)$, for the next-state prediction. Thus, updates are applied as follows:

$$V(\sigma) = V(\sigma) + \beta(\rho(t) + \gamma V(p(\sigma, \alpha, \mathrm{bg}(\sigma, \alpha))) - V(\sigma)) \tag{6}$$

These updates are applied to the state value learner XCS. The next section explains how V^* is approximated by XCS in interaction with ACS2. The implementation that stores all encountered situations explicitly and evaluates the situations according to equation 6 is referred to as ACS state value learner (ACS-SVL) in the remainder of this paper.

4.2 XCS as the State Value Learner

The learning classifier system XCS was introduced in Wilson (1995). XCS's learning is based on the accuracy of the reward prediction values of classifiers evolving a complete and accurate mapping of all possible situation action combinations in the environment. XCS was also used as a function approximator (Wilson, 2001). In our framework, XCS approximates the optimal state values V^* defined above. This section first gives an overview of the basic XCS structure and mechanisms. For further details the interested reader is referred to the cited literature as well as the recent algorithmic description of XCS (Butz & Wilson, 2001). In the following, it is explained how XCS interacts with ACS2 to evolve a proper function approximation.

XCS in a Nutshell Similarly to ACS2, XCS evolves a population of classifiers. In our framework, each XCS classifier has a condition part C_x with the same structure as conditions in ACS2's classifiers. Since XCS approximates state values in our framework and not values for state action combinations, the XCS classifiers have no action in this framework. Moreover, each XCS classifier has a reward prediction value p_x, a measure for the average error of the reward prediction value ϵ_x, and a fitness value F_x. The fitness in XCS measures the average accuracy of the classifier relative to the accuracies off all classifiers in the sets cl participates. Moreover, each classifier in XCS comprises a set size estimate ss_x that estimates the average match set size in which cl participates, the GA time stamp ts_x, an experience counter exp_x that counts the number of parameter updates the classifier underwent, and a numerosity value num_x that counts the number of actual micro-classifiers this "macroclassifier" represents.

In interaction with a problem, XCS encounters situations σ and forms match sets $[M]$ that consist of all classifiers in the population $[P]$ that match σ. Moreover, XCS encounters reinforcement values ρ_x and updates its parameters as follows:

$$\begin{aligned} p_x &= p_x + \beta_x(\rho_x - p_x) \\ \epsilon_x &= \epsilon_x + \beta_x(|p_x - \rho_x| - \epsilon_x) \\ ss_x &= ss_x + \beta_x(|[M]| - ss_x) \end{aligned} \tag{7}$$

The fitness is updated by first determining its absolute accuracy κ_x of the classifier by a scaled function and next, its relative accuracy κ'_x with respect to the current match set. Finally, its fitness value is updated according to this relative accuracy.

$$\kappa_x = \begin{cases} 1 & \text{if } \epsilon_x < \epsilon_{0_x} \\ \alpha_x \left(\frac{\epsilon_{0_x}}{\epsilon_x}\right)^{\nu_x} & \text{otherwise} \end{cases}$$
$$\kappa'_x = \kappa_x num_x / (\sum_{cl \in [M]} cl.\kappa_x cl.num_x)$$
$$F_x = F_x + \beta(\kappa'_x - F_x) \tag{8}$$

Parameter $\epsilon 0$ denotes an accuracy threshold. Once the average error ϵ_x is lower than this value the classifier is considered accurate. Parameter ν_x further scales the inaccurate classifiers and α_x adds an offset to further distinguish accurate from inaccurate classifiers.

After value updates and offspring generation by the ALP in the current match set $[M]$, a GA might be applied if the average last GA application in $[M]$ exceeds the threshold θ_{ga}. If a GA is applied, two classifiers are selected in $[M]$ usually applying proportionate selection based on fitness. In our implementation, we use tournament selection with a tournament size proportionate to the actual match set size $[M]$. The advantage of this selection method is subject of another paper. The selected classifiers are mutated with a niche mutation technique (see Butz & Wilson, 2001) that mutates an attribute in C_x with a probability μ_x. Moreover, the conditions of the two classifiers are crossed with a probability χ_x applying uniform crossover. The resulting offspring is added to the population. However, if there exists a more general, experienced ($\theta_{sub} < exp_x$), and accurate classifier ($\epsilon_0 > \epsilon_x$) in $[P]$, then an offspring classifier is not added but the numerosity of the subsuming classifier is increased by one. Moreover, classifiers are deleted in the population if the population size exceeds the threshold N_x. For deletion, classifiers are selected using proportionate selection with respect to the set size estimate of the classifiers, their experience, and their fitness.

XCS Interacting with ACS2 For the interaction with ACS2, it is necessary to define when and how parameters are updated in XCS. Each time, ACS2 perceives a situation σ and an immediate reward ρ, XCS is called.

The reinforcement value for reward prediction updates in XCS are determined by a combination of the current predictive model knowledge of ACS2 and the current state values of XCS.

$$\rho_x = \rho + \gamma \max_\alpha \frac{\sum_{\{cl|cl \text{ matches } P(\sigma,\alpha,\text{bg}(\sigma,\alpha))} cl.p_x \cdot cl.\kappa \cdot cl.num_x}{\sum_{\{cl|cl \text{ matches } P(\sigma,\alpha,\text{bg}(\sigma,\alpha))} cl.\kappa \cdot cl.num_x} \tag{9}$$

This is similar to the prediction array calculation in XCS but uses the predictive model of ACS to determine next states. Moreover, the reward estimates are weighted according to the current accuracy instead of the current fitness of a classifier. The equation essentially adds the current immediate reward to the best state value prediction in all possible future state.

Once ρ_x is determined, all classifiers in the current match set $[M]$ of XCS can be updated according to the update rules defined in equations 7 and 8. Moreover, the GA application in XCS can be invoked.

Restricted and Additional State Value Evaluation First experiments showed that XCS is not able to evolve a complete state value representation reliably. The problem is that XCS evolves classifiers with very general conditions, if the position in which immediate reward is experienced is not visited early enough nor frequently enough, recovery from this early over-generalization effect appears to be hard. Thus, two mechanisms are added to the XCS evaluation procedure. (1) XCS parameter updates and GA application are restricted. (2) An additional list of states is maintained on which internal evaluations are performed. Both mechanisms are explained in further detail below.

To prevent early over-generalization and disruption due to strongly biased sampling, learning is only triggered in XCS, if there exist useful prediction values. Moreover, learning is never triggered more than once in a row. Prediction values are considered useful if ACS2 is actually able to predict at least one consequent state and XCS has a state value representation of the predicted states. To prevent XCS from updating classifiers in one situation multiple times in a row, the last situation in which an update occurred is remembered. If the same situation is encountered in the next step, neither parameter updates nor genetic algorithm are applied.

Additionally, XCS keeps a list of states of size sl_x in which high average parameter updates occurred. Each entry in the list stores a perceived situation together with its average error in the state as well as the encountered prediction value ρ_x. After each actual parameter update and GA application, the state list is scanned. If the current state exists in the state list, the prediction value and average error value are updated. The entry will be deleted once the average error decreases below $10 * \epsilon_0$. If the current state does not exist and the lowest average error in the list is smaller than the one encountered, the entry is replaced by the actual situation with corresponding values. Next, an internal update is triggered in which an entry is chosen from the maintained list proportional to the average error values. The update is applied using the stored ρ_x value. Next, the average error is replaced by the average error of the current update. Moreover, a GA might be triggered.

4.3 Resulting Behavior

With the definitions above, we have constructed a learning agent that learns a predictive model of its environment (i.e. the state transition function in an MDP) as well as the optimal state value function of the problem. The state value function is learned by using the model predictions of ACS2 and the value estimates in the predicted situations to update the reward predictions in XCS. In the remainder of this work, we will term the combination of ACS2 and XCS the extended anticipatory classifier system XACS.

What remains is to show how anticipatory behavior is applied in XACS. Given a current situation σ we defined the optimal policy π^* above given the state transition function f and the optimal state value function V^*. Since ACS2 learns the state transition function and XCS the optimal state value function, an optimal policy can now be defined by

$$\pi(\sigma) = \arg\max_{\alpha} \frac{\sum_{\{cl|cl \text{ matches } P(\sigma,\alpha,\text{bg}(\sigma,\alpha))\}} cl.p_x \cdot cl.\kappa \cdot cl.num_x}{\sum_{\{cl|cl \text{ matches } P(\sigma,\alpha,\text{bg}(\sigma,\alpha))\}} cl.\kappa \cdot cl.num_x} \tag{10}$$

since $f(\sigma, \alpha)$ is set equal to the prediction done by the best guess of ACS2 and $V^*(\sigma)$ is set equal to the accuracy weighted average prediction of classifier in XCS that match σ.

Essentially, this policy is an anticipatory policy since it predicts the result of each action before action execution and then chooses the best action with respect to the prediction. The major difference to traditional dynamic programming (DP) approaches is that the predictive model as well as the state values are learned online and generalized online.

4.4 Performance in the Blocks World

In order to evaluate the performance of XACS, we monitor its performance in the blocks world introduced above. XACS interacts with the blocks world by the provided s gripping and releasing actions. A reinforcement of 1000 is provided when the first stack reaches its specified height y. Moreover, the environment is reset to a random situation in this case. The experiments are divided into exploration and exploitation trials. During exploration, actions are chosen at random applying the biased exploration procedure published in (Butz, 2002b) with a probability of p_b. All learning components are applied. Exploit trials are used for performance evaluation purposes only and consequently no parameter updates nor offspring generation is triggered. Actions are chosen according to policy $\pi(\sigma)$ defined in equation 10. A trial is limited to 50 steps in all the experiments. If the goal was not reached after 50 steps, the environment is reset to a random situation. Performance is displayed by averaging the steps to the goal over the last 50 exploitation trials. Curves are averaged over fifty runs. [1]

The left hand side of figure 3 shows that XACS is able to adapt its behavior in the $s = 4$, $b = 4$, $y = 4$ blocks world reaching optimal behavior after approximately 150 learning trials. The figure also displays performance of ACS2 clearly showing that ACS2 is not able to evolve optimal behavior in the problem. The right hand side of figure 3 compares performance of XACS with performance of

[1] If not stated differently, parameters were set to: $\beta = 0.05$, $u_{max} = \infty$, $\gamma = 0.9$, $\theta_{ga} = 100$, $\mu = 0.3$, $\chi = 0.8$, $\theta_{as} = 20$, $\theta_{exp} = 20$, $\epsilon = 1$, and $p_b = 0.5$. XCS parameters were set to $N_x = 2000$, $\beta_x = 0.05$, $\alpha_x = 1$, $\epsilon_0 = 1$, $\nu_x = 5$, $\theta_{GA_x} = 25$, $\chi_x = 0.8$, $\mu_x = 0.01$, $\theta_{del_x} = 20$, $\delta_x = 0.1$, $\theta_{sub_x} = 20$, $p_{I_x} = 0$, $\epsilon_{I_x} = 10$, $f_{I_x} = 0$. Note that some of the parameters were not introduced above but are provided for completeness. The meaning of the parameters can be inferred from the cited literature on ACS2 and XCS.

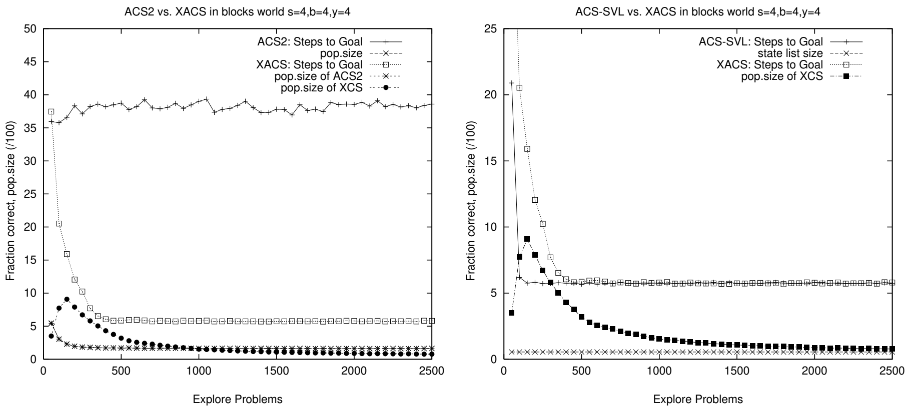

Fig. 3. As observable on the left hand side, ACS2 suffers from model aliasing while XACS is able to learn an optimal policy in the blocks world problem. XACS achieves similar performance to the state value learner without generalization (right hand side).

ACS-SVL in which the state value learner does not generalize but keeps a list of all encountered states, propagating reinforcement similar to XACS over the state list. The comparison shows that XACS eventually reaches the goal as fast as ACS-SVL but learning takes slightly longer. The population size of the XCS module in XACS approaches the actual number of states in the problem. This shows that the generalization mechanisms does not cause disruption but it does not result in any gain, either.

To see if the generalization mechanism was functioning at all, we took a look at the evolving classifier list of XCS. Table 4.4 lists the highest numerosity classifiers in the blocks world problem $s = 4$, $b = 4$, $y = 4$. The condition part codes the four positions in each of the four stacks starting from the lowest position in each stack. Also the gripper state is coded. It can be seen that all classifiers are accurate and predict the correct state value. Moreover, nearly all classifiers are maximally general except for classifier number 9,10, and 12. Classifier 9's specified problem subspace is more specific than the one of classifier 11 and should be eventually absorbed by it. Similarly, classifier 10 is more specific than classifier 7 and classifier 12 is more specific than classifier 8. Note that classifiers 9 and 10 are more specific in the BW problem than 11 and 7 although they specify less attributes. The listed classifiers in table 4.4 confirm that XCS successfully evolves a generalized state-value representation.

The left hand side of figure 4 shows the behavior of XACS in the larger blocks world problem where $s = 7$, $b = 7$, $y = 4$. In this problem setting, XACS actually evolves a state value representation with a smaller number of classifiers than states in the problem confirming that the generalization mechanism works properly. Moreover, XACS reaches the optimal policy faster than ACS-SVL. Applying XACS to the blocks world problem of size $s = 8$, $b = 8$, and $y = 4$, displayed on the right hand side of figure 4, shows that ACS-SVL fails to evolve

	C_x	p_x	ϵ_x	F_x	num_x	exp_x	ss_x
1	.### #### #### #### A	478	0.001	0.992	199	27880	210
2	.### #### #### #### .	430	0.001	0.972	189	20625	204
3	A.## #### #### #### .	531	0.002	0.998	167	15695	181
4	###A #### #### #### #	1000	0.000	0.851	138	1565	187
5	A.## #### #### #### A	590	0.001	0.820	136	12964	180
6	##A# #### #### #### A	900	0.000	1.000	123	1582	157
7	#A.# #### #### #### .	656	0.004	0.803	95	8328	145
8	#A.# #### #### #### A	729	0.000	0.643	90	5750	169
9	##A# #### #### A### #	810	0.000	0.538	88	243	177
10	#A## #### #### #A## #	656	0.004	1.000	57	351	174
11	##A. #### #### #### .	810	0.000	0.638	57	4100	131
12	#A## A### #### #### A	729	0.001	1.000	53	1182	175
	...						

Table 1. A typical classifier list ordered by their numerosity values show that XCS properly evolves generalized state values in the blocks world problem.

an optimal policy during the 2500 trials due to the large state space. XACS does evolve an optimal policy with a state-value representation of less than 720 classifiers—much smaller than the actual 9867 states possible in the problem which are all specified by the explicit state value learner.

The results confirm that the combination of an online generalizing predictive model learner, represented by ACS2, with a state value function approximation method, implemented with the XCS mechanism, was successful. While it was shown in (Butz, 2002a) that ACS2 is able to build a compact model of the underlying state transition function in the blocks world problem, the results herein show that it is also possible to learn a generalized representation of a state value function yielding improved performance, a compact representation, as well as anticipatory behavior. The next section puts the results in a broader perspective and compares the resulting system with other existing approaches.

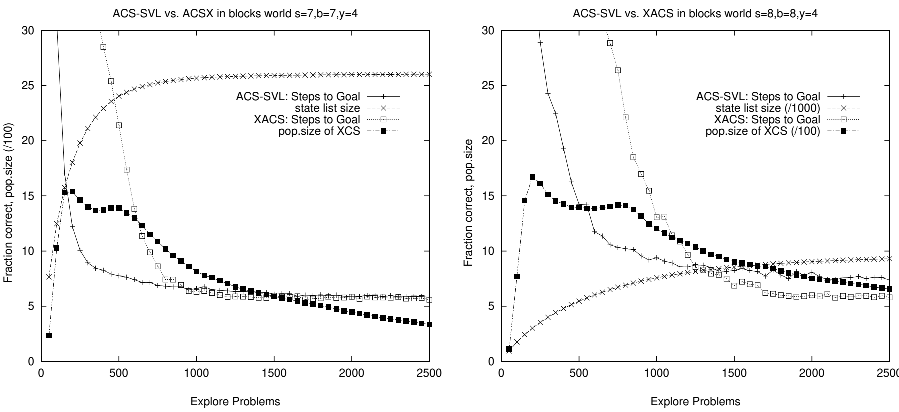

Fig. 4. Looking at larger blocks worlds, XACS outperforms ACS2 with a non-generalizing state value learner memory wise as well as policy wise.

5 Related Systems

The previous section confirmed that a combination of a generalizing predictive model learner with a generalizing state value learner can yield optimal performance with a more compact representation. This section focuses on the relation of this approach to other anticipatory learning systems.

As explained above, XACS essentially learns the state transition function of an MDP and evolves a state value representation of a Bellman equation that gives rise to an optimal behavioral policy.

The basic underlying concept of evolving an optimal state value representation is that of dynamic programming (DP) in which a state value representation is usually generated iterating over all possible states of a problem. DP usually assumes that state transition function as well as payoff function are known beforehand. Thus, DP only learns the state value function. XACS, on the other hand, learns the state transition function, the payoff function, and the state value function.

The dynamical reinforcement architecture Dyna (Sutton, 1990) is a general framework in which the state transition function as well as the payoff function is learned online. Dyna implementations represent optimal policies by reinforcement values for state-action combinations or state values. Most Dyna implementations do not generalize online state transition function nor state-value function.

More closely related are two other anticipatory learning systems that take a similar approach to XACS: (1) Yet Another Classifier System (YACS) (Gérard & Sigaud, 2001b; Gérard & Sigaud, 2001a) and (2) the modular anticipatory classifier system (MACS) (Gérard, Meyer, & Sigaud, 2002). Both systems learn the state transition function of a problem similar to ACS2. YACS basically has the same predictive model representation as ACS2 but uses more informed mechanisms to learn the model. MACS evolves separate predictive rules for each perceptual attribute being able to exploit additional regularities in some environments. Both approaches keep a list of all encountered states using DP techniques similar to the one in ACS-SVL to evolve an optimal policy represented in state values for each state in the list. YACS represents immediate reward directly in the predictive rules which could result in a model aliasing problem for immediate reward predictions. MACS evolves identical state values to the ones in ACS-SVL and XACS. Neither YACS nor MACS attempt to approximate the state values with a function approximation mechanism.

6 Anticipatory Behavior

While we showed above that the basic behavior in XACS is anticipatory, there are many ways to improve and enhance the possibilities of XACS by implementing further anticipatory mechanisms. Several of these possibilities are discussed in this section.

6.1 Task-dependent Attention

The generalization in the state value function approximation mechanism could directly be used to implement task-dependent attentional mechanisms in XACS. The basic theory of attentional mechanisms distinguishes early and late processing stages framing the attentional constraints in a controlled parallel framework (Pashler, 1998). It is theorized that early perceptual mechanisms work in parallel extracting for example simple perceptual properties such as color or edges in the visual field. Late mechanisms usually undergo a bottleneck and cannot be processed in parallel. An attentional mechanism decides which environmental properties are processed further.

A computational approach to attention was proposed using a saliency map in which the salience of distinct perceptual properties is combined (Koch & Ullmann, 1985). A winner-takes-all algorithm determines the location that will be processed further. Higher cortical influences on the winner-takes-all algorithm are suggested.

Further influences on the saliency map could for example be implemented by the task-dependent importance of predicted future attributes resulting in a sort of preparatory attention. In XACS the generalized situations for the prediction of accurate state values could influence the actual matching and decision making process. Since the specified attributes in the generalized situations essentially denote the task relevant attributes, classifier selection could be restricted to those classifiers that predict perceptual changes in task relevant attributes. Moreover, task irrelevant perceptual attributes could be disregarded from the matching process in ACS2. Predictions could be restricted to the task relevant attributes resulting in further processing speed-ups.

The attentional mechanism would be an anticipatory mechanism in that the choice of current attention is influenced by the predicted relevance of attributes.

6.2 Multi-Task and Attention

Right now, the attentional mechanism would be restricted to the only available task in the current environment (i.e. maximization of reinforcement). However, multiple tasks are imaginable in XACS with separate XCS state approximation modules for each of the tasks. For example, XACS could be modeled to continuously find food and water in an environment. If the current need of the animat was to get food (because it is hungry), the animat's policy would be based on the state values in the 'hunger' XCS module.

To accomplish multi tasks, either the environment would need to provide a reward vector or reinforcement would be generated internally once a particular need was satisfied. The preparatory attentional mechanisms would consequently depend on the specialized attributes in the XCS module for the current strongest need. For example, if the animat was currently hungry, specifications in the 'hunger' XCS module would be considered whereas if it was thirsty, it could bias the mechanisms on the specifications in the 'thirst' XCS module. Interactions of hunger and thirst would need to be coordinated in an additional module which could be termed a "motivation coordinator".

6.3 Interactive Learning Mechanisms

Right now the two distinct learning modules XCS and ACS2 do not exchange any information about the importance or relevance of attributes. However, in natural environments insights in the generalization possibilities for the predictive model should also be useful for the state value learner module and vice versa. Thus, the learning mechanisms in XCS as well as ACS2 could be biased toward the insights gained in the other model. For example, the specialization mechanisms in ACS2 could be biased toward specifying usually task-important attributes or genetic generalization could be biased toward usually task-independent attributes. Mutation in XCS could be biased in a similar way. This interaction, however, would rather result in implicit anticipatory mechanisms in the learning component, than in the use of explicit anticipations before action selection.

6.4 Internal Policy Updates

Finally, it is necessary to mention the possibility of applying further internal updates to the state value learner XCS. The currently maintained state list can be compared to the prioritized sweeping approach in Dyna (Moore & Atkeson, 1993). Beyond single-state updates, it could be useful to extend updates to multiple steps consequently reaching a larger neighborhood. Hereby, the update depth could be made dependent on the surprise level of the stored state represented by the average error value recorded for each attribute. Moreover, it could be useful to apply random updates in the XCS module as previously done in the original Dyna implementation (Sutton, 1990) or in an application of ACS (Stolzmann, Butz, Hoffmann, & Goldberg, 2000) to psychological learning experiments.

7 Summary and Conclusions

This paper has introduced a function approximation module to ACS2 that approximates the state value function suitable to learn an optimal policy in combination with the model learning component ACS2. The state value learner was implemented by a modified version of XCS. The resulting system XACS was tested on a blocks-world problem in which it proved to outperform the previous policy learning mechanism in ACS2. It was also shown that non-generalizing state value mechanisms can be outperformed in memory requirements as well as in behavioral performance. Moreover, it was shown that the resulting system is essentially an anticipatory behavior system predicting action consequences and biasing current action choice on the predictions.

Although the presented results point to a promising direction it has to be noted that the state generalizing mechanism does not seem to be robust as applications in other MDPs have shown. The problem is related to the problem space sampling and resulting over-generalization problems in the XCS module. Since naive function approximation mechanisms do not work for the approximation of a state value function in general, it has been suggested that it is necessary to constrain the function approximation further on robustness during DP or explicitly prevent divergence (Boyan & Moore, 1995). The accuracy based fitness approach in XCS appears to be not enough to constrain the value function approximation. Research is in progress to further analyze and solve the problem in XCS and XACS.

Acknowledgments The authors would like to thank Xavier Llora, Martin Pelikan, and Kumara Sastry as well as the whole IlliGAL lab for fruitful discussions and useful comments.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The work was also funded by the German Research Foundation (DFG) under grant HO1301/4.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems*, *7*.

Butz, M. V. (2002a). *Anticipatory learning classifier systems.* Boston, MA: Kluwer Academic Publishers.

Butz, M. V. (2002b). Biasing exploration in an anticipatory learning classifier system. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: 4th International Workshop, IWLCS 2001* (pp. 3–22). Berlin Heidelberg: Springer-Verlag.

Butz, M. V., & Wilson, S. W. (2001). An algorithmic description of XCS. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: Third International Workshop, IWLCS 2000* (pp. 253–272). Berlin Heidelberg: Springer-Verlag.

Gérard, P., Meyer, J.-A., & Sigaud, O. (submitted, 2002). Combining latent learning and dynamic programming in MACS. *European Journal of Operational Research*.

Gérard, P., & Sigaud, O. (2001a). Adding a generalization mechanism to YACS. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, 951–957.

Gérard, P., & Sigaud, O. (2001b). YACS: Combining dynamic programming with generalization in classifier systems. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: Third International Workshop, IWLCS 2000* (pp. 52–69). Berlin Heidelberg: Springer-Verlag.

Hoffmann, J. (1993). *Vorhersage und Erkenntnis: Die Funktion von Antizipationen in der menschlichen Verhaltenssteuerung und Wahrnehmung. [Anticipation and cognition: The function of anticipations in human behavioral control and perception.].* Göttingen, Germany: Hogrefe.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–258.

Koch, C., & Ullmann, S. (1985). Shifts in selective attention: Towards the underlying neural circuitry. *Human Neurobiology*, *4*, 219–227.

Moore, A. W., & Atkeson, C. (1993). Prioritizes sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, *13*, 103–130.

Pashler, H. E. (1998). *The psychology of attention.* Cambridge, MA: MIT Press.

Stolzmann, W. (1998). Anticipatory classifier systems. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 658–664.

Stolzmann, W. (2000). An introduction to anticipatory classifier systems. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Learning Classifier Systems: From Foundations to Applications* (pp. 175–194). Berlin Heidelberg: Springer-Verlag.

Stolzmann, W., Butz, M. V., Hoffmann, J., & Goldberg, D. E. (2000). First cognitive capabilities in the anticipatory classifier system. *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, 287–296.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 216–224.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, Cambridge, UK.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S. W. (2001). Function approximation with a classifier system. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, 974–981.