A Bayesian Approach to Learning Classifier Systems in Uncertain Environments

Davide Aliprandi
Politecnico di Milano
Department of Electronics and Information, via Ponzio 34/5, 20133, Milan
da666404@lau.polimi.it

Alex Mancastroppa
Politecnico di Milano
Department of Electronics and Information, via Ponzio 34/5, 20133, Milan
am666416@lau.polimi.it

Matteo Matteucci
Politecnico di Milano
Department of Electronics and Information, via Ponzio 34/5, 20133, Milan
matteucc@elet.polimi.it

ABSTRACT

In this paper we propose a Bayesian framework for XCS [9], called BXCS. Following [4], we use probability distributions to represent the uncertainty over the classifier estimates of payoff. A novel interpretation of classifier and an extension of the accuracy concept are presented. The probabilistic approach is aimed at increasing XCS learning capabilities and tendency to evolve accurate, maximally general classifiers, especially when uncertainty affects the environment or the reward function. We show that BXCS can approximate optimal solutions in stochastic environments with a high level of uncertainty.

Categories and Subject Descriptors: I.2.6 [Computing Methodologies]: Artificial Intelligence – *Learning*

General Terms: Algorithms, Experimentation, Theory.

Keywords: Bayesian Q-Learning, Learning Classifier Systems, XCS, Exploration Strategy, Value of Information.

1. INTRODUCTION

Reinforcement learning in Markov Decision Processes (MDP) is the framework where an agent learns how to optimally act by interacting with a stochastic environment. Q-Learning [8] is the most known reinforcement learning algorithm proved to eventually converge to the optimal policy in its tabular implementation (i.e., using a completely specified representation of state-action values), if all possible states are visited infinitely often and two simple conditions on the learning rate hold.

Learning Classifier Systems use a rule base of classifiers to approximate the tabular representation of Q-Learning obtaining a twofold result: (i) using generalization it is possible to represent the Q-table in a compact way facing the curse of dimensionality (ii) a state-action value is approximated by a set of classifiers resulting in a better approximation w.r.t. the simple state aggregation technique.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

XCS [9] represents a milestone in learning classifier system research because of the use of Q-Learning in classifier payoff evaluation and its generalization mechanism based on *accuracy*, which is a measure of the degree of *overgeneralization* [9]. In deterministic and completely observable environments, overgeneralization is the unique source of inaccuracy; while, considering more general environments, there are two other sources of inaccuracy: *perceptual aliasing* and *stochastic transitions*. When experiencing perceptual aliasing, the agent acts in environments that are partially observable by its sensors; in such cases the agent is not always able to determine the best action only looking at its sensors because two or more situations may be aliased, i.e., perceived as the same. The latter, and probably most common, source of inaccuracy is found in stochastic environments, where the effects of agent actions may be uncertain [7] as well as the reward function.

In Q-Learning, the effects of these three sources of inaccuracy are faced at the same time, but XCS is able to single out the inaccuracy due to overgeneralization only when the environment is both completely observable and deterministic. Lanzi [5] showed that when the environment is partially observable, memory can be added to XCS in order to disambiguate perceptually aliased situations. Besides, Lanzi and Colombetti [7] introduced an extension of XCS, called XCSμ, that tries to separate the inaccuracy due to overgeneralization from that due to uncertainty and showed that the system converges to an optimal solution for a degree of uncertainty up to 0.5, i.e., when the results of an action sorts the desired effect only once out of two.

In XCS, as with Q-Learning, the exploration strategy the agent uses while acting in the environment is a fundamental aspect of learning. A good exploration strategy should balance the expected gain from exploration and the cost of trying possibly suboptimal actions when better ones are available to be exploited. Finding optimal solution of the exploration/exploitation trade-off requires solving an MDP over information states*. The aim is to find a policy for the agent that maximizes its expected reward; although this problem is well-defined, given a prior distribution over possible environments, it is not easy to solve exactly. From a naive perspective, we would like to have an informed explo-

*The information state is the set of all possible probability distributions over environment models that can be reached by executing all possible action sequences and receiving any possible percept sequence and reward sequence.

ration strategy able to explore if there is some *information gain*, i.e., when the agent improves either its policy or its knowledge, otherwise we would like to exploit the actual rule base to refine payoff estimates.

In this paper, we propose an evolution of XCS, called BXCS (Bayesian XCS), based on Bayesian probability distributions over classifier estimates of payoff, aimed at facing the environment uncertainty. Broadly speaking, we implement a Bayesian Q-Learning algorithm instead of classical Q-Learning to evaluate classifier payoff. Moreover, we define a new fitness function in order to take directly into account the uncertainty of the environment during the exploration phase and focus on exploration actions that have a higher information gain. We show that BXCS can learn better solutions than classical XCS when the degree of uncertainty that affects the environment becomes higher. Our main claim is that the Bayesian fitness function can take into account the uncertainty on the effects of agent's actions in a stochastic environment better than a non-probabilistic one and does not affect the XCS performances in deterministic ones.

This paper is organized as follows. In Section 2, we provide a short overview of XCS, with all details that are important for the remainder of the paper. In Section 3, we describe Bayesian Q-Learning techniques and Section 4 reports all the relevant modifications to apply them with XCS. Finally, we test the new system and show the results of a number of experiments in Section 5. All experiments in this paper were carried out with Lanzi's `xcslib` [6]. Source code and configuration files required to reproduce the results are available on request.

2. XCS CLASSIFIER SYSTEMS

In this section, we give a short description of XCS following [1]. For a complete description, we refer the interested reader to the original paper by Wilson [9], and to the recent algorithmic description in [2].

XCS is a classifier system which differs from the one firstly introduced by Holland mainly because (i) it has a very simple architecture, (ii) there is no message list, and, most important, (iii) the traditional strength is replaced by three different parameters: prediction, prediction error and accuracy.

XCS acts as the classical reinforcement learning agent [8]: it receives inputs describing the current state of the environment, reacts executing actions, and eventually receives a reward (in the form of a real number) as an indication of the goodness of its actions. The goal is to maximize the total reward collected in the long run. XCS achieves this by learning an *action-value* function which maps each *state-action* pair into a real number, called *payoff*, analogous to the Q-values in Q-Learning.

Classifier parameters. Classifiers in XCS consist of a condition $C \in \{0, 1, \#\}^L$ (# is a "don't care" symbol, L is the length of the input string) that specifies which input states the classifier *matches*, and an associated action a. Moreover, to each classifier are associated three parameters: (i) the *prediction* p, which estimates the payoff that the system expects if the classifier is used, (ii) the *prediction error* ε, which estimates the error of the prediction p, and (iii) the *fitness* F which estimates the accuracy of the payoff prediction p.

It is worth noticing that, because of the generalization mechanism, in XCS the expected reward of a state-action pair is estimated by a mixture of classifiers that differ for generality. This mixture of classifiers is optimized by the genetic component using the classifier fitness as target.

Performance component. At each time step, the system input is used to build the *match set* $[M]$ containing the classifiers in the population whose condition part is satisfied by the current sensory configuration. If the match set contains less than θ_{mna} classifiers, new classifiers are created through *covering*, with the condition part matching the current state and random action. For each action a in the match set the system prediction $P(a)$ is computed as the weighted average of the classifier predictions that advocate the action a using classifier fitness as weight. The value $P(a)$ gives an evaluation of the expected payoff if a is performed.

Action selection strategy can be either *deterministic*, i.e., the action with the highest prediction is chosen (exploitation), or *probabilistic*, i.e., the action is chosen with a certain probability (exploration). The classifiers in $[M]$ which propose the selected action form the so called *action set* $[A]$ and they will be possibly rewarded. The selected action is then performed in the environment and a scalar reward r is returned to the system together with a new input configuration.

Reinforcement component. The reward received from the environment is used to update the parameters of the classifiers in the action set. First, the prediction is updated:

$$p \leftarrow p + \eta(P - p), \quad 0 < \eta \leq 1,$$

where η denotes the *learning rate*, and P is computed as the sum of the reward received at previous time step and the maximum system prediction, discounted by a factor γ ($0 \leq \gamma < 1$). Next, the prediction error is adjusted using the delta rule formula:

$$\varepsilon \leftarrow \varepsilon + \eta(|P - p| - \varepsilon).$$

Updating fitness is slightly more complex: initially, the prediction error is used to evaluate the classifier accuracy k as $k = \alpha(\varepsilon/\varepsilon_0)^{-\nu}$ if $\varepsilon > \varepsilon_0$ or $k = 1$ otherwise. Subsequently, the relative accuracy k' of each classifier is computed from k and, finally, the fitness parameter is adjusted by the rule:

$$F \leftarrow F + \eta(k' - F).$$

Discovery component. In XCS a genetic algorithm (GA) is applied to the classifiers in the current action set when the average time since the last application exceeds a threshold θ_{ga}. The GA selects two classifiers with probability proportional to their fitness, and applies on them crossover and mutation operators respectively with probability p_χ and p_μ.

3. BAYESIAN Q-LEARNING

Bayesian Q-Learning was introduced by Dearden et al. in [4] to face uncertainty of the environment by means of a Bayesian probabilistic approach. In the Bayesian Q-Learning framework, we need to consider prior distributions over Q-values, and then update these priors based on the reward collected by the agent acting in the environment. Formally, let $R_{s,a}$ be a random variable that denotes the *total* discounted reward received when action a is executed in state

s and an optimal policy is followed thereafter. Initially, we are uncertain about how $R_{s,a}$ is distributed and this is represented by the initial prior; in particular, we want to learn the value $Q^*(s,a) = E[R_{s,a}]$.

To make the approach practical, the following assumptions are taken:

Assumption 1: $R_{s,a}$ has a normal distribution; this implies that to model the uncertainty about the distribution of $R_{s,a}$ it is possible to use a distribution over the *mean* $\mu_{s,a}$ and the *precision* $\tau_{s,a}$ of $R_{s,a}$ (note that the precision of a normal variable is the inverse of its variance, i.e., $\tau_{s,a}$=$1/\sigma^2_{s,a}$);

Assumption 2: The prior distribution over $\mu_{s,a}$ and $\tau_{s,a}$ is independent of the prior distribution of $\mu_{s',a'}$ and $\tau_{s',a'}$ for $s \neq s'$ or $a \neq a'$;

Assumption 3: The prior is a *Normal-Gamma* distribution: $p(\mu_{s,a}, \tau_{s,a}) \sim NG(\mu_0, \lambda, \alpha, \beta)$; the relationship between this bivariate distribution and the random variable $R_{s,a}$ is: $\tau \sim Gamma(\alpha, \beta)$ [†], and $\mu \mid \tau \sim N(\mu_0, 1/(\lambda\tau))$[‡] being *Gamma* and N respectively the Gamma and Normal distributions [3];

Assumption 4: At any stage, the agent's posterior over $\mu_{s,a}$ and $\tau_{s,a}$ is independent of the posterior $\mu_{s',a'}$ and $\tau_{s',a'}$ for $s \neq s'$ or $a \neq a'$.

Through assumptions 2 and 3 we can represent our prior by a tuple of *hyperparameters* for each state s and action a. Assumption 4 is likely to be violated in an MDP; nonetheless, we shall assume that we can represent the posterior as though the observations were independent. To update prior distribution after receiving a new sample from $R_{s,a}$ we apply the result of the following theorem:

THEOREM 1. *Let $p(\mu,\tau) \sim NG(\mu_0, \lambda, \alpha, \beta)$ be a prior distribution over the unknown parameters μ and τ for a normally distributed variable R (with mean μ and variance $1/\tau$), and let $r_1, \cdots, r_n$ be n independent samples of R with first two moments:*

$$M_1 = \frac{1}{n}\sum_i r_i, \quad \textit{and} \quad M_2 = \frac{1}{n}\sum_i (r_i - M_1)^2.$$

Then $p(\mu,\tau \mid r_1, \cdots, r_n) \sim NG(\mu_0', \lambda', \alpha', \beta')$ where:

$$\begin{aligned} \mu_0' &= \frac{\lambda\mu_0 + nM_1}{\lambda + n}, \quad \lambda' = \lambda + n, \quad \alpha' = \alpha + \frac{1}{2}n, \\ \beta' &= \beta + \frac{1}{2}n(M_2 - M_1^2) + \frac{n\lambda(M_1 - \mu_0)^2}{2(\lambda + n)}. \end{aligned}$$

That is, given a Normal-Gamma prior, the posterior after any sequence of independent observations is again a Normal-Gamma distribution. For more details about the aforementioned assumptions and Theorem 1 refer to [4].

4. BXCS CLASSIFIER SYSTEMS

We consider the Bayesian Q-Learning technique introduced in the previous section to represent the uncertainty a classifier has about its estimate of Q-value (i.e., predicted reward). By keeping and propagating distributions over Q-value, rather than point estimates (as in traditional Q-Learning), we can take more informed decisions aimed at an informed global exploration.

4.1 Classifier: a new interpretation

We are interested in exploiting the Bayesian Q-Learning concepts to define probability distributions over the predicted payoff in order to assess the classifier uncertainty. In doing this, a classifier can be considered as an entity able to act by taking directly into account the uncertainty of the environment and the uncertainty in reward perception due to generalization, i.e., the uncertainty in representing the reward over a portion of the state-action space that aggregates more than one possible configurations.

From the architectural point of view, the traditional parameters used in Wilson's XCS [9] are replaced by a tuple $< \mu_0, \lambda, \alpha, \beta >$ of hyperparameters which define a posterior distribution $p(\mu,\tau)$ of the random variable R, the total discounted reward received when the classifier matches the state and its action is chosen by the system.

The three main parameters of XCS, i.e. prediction, prediction error and accuracy, are thus replaced by their probabilistic counterparts, i.e. μ_0, λ, α, β introduced above, and the new *Bayesian accuracy* k_B (see Section 4.4). These parameters characterize the uncertainty about the environment and reward perception. This is an evolution of XCS, and in particular of its concept of accuracy.

4.2 Prediction array and action selection

For each action a in $[M]$, BXCS computes the *action value* $P(a)$, which represents an estimate of the payoff that the system expects to receive when action a is performed. $P(a)$ is computed in two steps: (i) we define a joint distribution over a from the posterior of all matching classifiers that propose action a. Then, (ii) we apply to this joint distribution one of the Bayesian action selection strategies proposed in the next section. The different values of $P(a)$ form the *prediction array*. BXCS selects an action w.r.t. the values in the prediction array.

To get the *action value* $P(a)$ for action a we need to follow these four steps:

1. for each classifier cl_i in $[M]_a$ compute R_i as:
$$\begin{aligned} \mu_i &= \mu_{0_i}, \\ \sigma_i^2 &= \frac{\lambda_i + 1}{\lambda_i}\frac{\beta_i}{\alpha_i - 1}. \end{aligned}$$
being $p(\mu_i, \tau_i) \sim \mathcal{NG}(\mu_{0_i}, \lambda_i, \alpha_i, \beta_i)$ the prior defined by cl_i on $R_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$;

2. from this derive $R_a \sim \mathcal{N}(\mu_a, \sigma_a^2)$ with:
$$\begin{aligned} \mu_a &= \sum_{cl_i \in [M]_a} F_i' \mu_{0_i}, \\ \sigma_a^2 &= \sum_{cl_i \in [M]_a} F_i'^2 \sigma_i^2, \\ F_i' &= \frac{F_i}{\sum_{cl_i \in [M]_a} F_i}; \end{aligned}$$

3. the joint Normal-Gamma $p(\mu_a, \tau_a) \sim \mathcal{NG}(\mu_{0_a}, \lambda_a, \alpha_a, \beta_a)$

[†]For τ we have: $E[\tau] = \frac{\alpha}{\beta}$, and $var[\tau] = \frac{\alpha}{\beta^2}$.

[‡]For μ we have: $E[\mu] = \mu_0$, and $var[\mu] = \frac{\beta}{\lambda(\alpha-1)}$.

is obtained by:

$$\begin{aligned}\mu_{0_a} &= \mu_a,\\ \lambda_a &= \sum_{cl_i \in [M]_a} F_i' \lambda_i,\\ \alpha_a &= \sum_{cl_i \in [M]_a} F_i' \alpha_i,\\ \beta_a &= \sigma_a^2 \frac{\lambda_a(\alpha_a - 1)}{\lambda_a + 1};\end{aligned}$$

4. compute $P(a)$ using the appropriate selection strategy.

The selection strategy is crucial to the performance of the algorithm. The basic idea is balancing *exploration* of untested actions and *exploitation* of actions that are known to be good. Applying a *greedy selection* we would select the action a that maximizes the expected reward value $E[\mu_a]$. Unfortunately, selecting the action with the greatest mean would reduce the attempts to perform exploration and does not take into account any uncertainty about the predicted payoff. In the following section we present two alternative approaches for exploration directly derived from the application of Bayesian Q-learning: *Q-value sampling* and *Myopic-VPI* [4].

Q-value sampling. In Q-value sampling, we select actions stochastically, based on the system current belief of their optimality. In other words, action a is performed with probability given by:

$$\begin{aligned}p(a = \arg\max_{a'} \mu_{0_{a'}}) &= p(\forall a' \neq a, \mu_{0_a} > \mu_{0_{a'}})\\ &= \int_{-\infty}^{+\infty} p(\mu_{0_a} = x) \prod_{a' \neq a} p(\mu_{0_{a'}} < x) dx. \qquad (1)\end{aligned}$$

Note that if $p(\mu_a, \tau_a) \sim NG(\mu_{0_a}, \lambda_a, \alpha_a, \beta_a)$, then the marginal density of μ_a is:

$$\begin{aligned}p(\mu_a) &= \left(\frac{\lambda_a}{2\pi}\right)^{\frac{1}{2}} {\beta_a}^{\alpha_a} \gamma_a \left(\beta_a + \frac{\lambda_a}{2}(\mu_a - \mu_{0_a})^2\right)^{\delta_a}, \qquad (2)\\ \gamma_a &\doteq \frac{\Gamma(\alpha_a + \frac{1}{2})}{\Gamma(\alpha_a)}, \quad \textit{and} \quad \delta_a \doteq -(\alpha_a + \frac{1}{2}),\end{aligned}$$

where Γ is the Gamma function. In practice, we can avoid the computation of Equation 1. Instead, for each action a we sample a value from the corresponding $p(\mu_a)$ and execute the action with the highest sampled value. The cumulative distribution of $p(\mu_a)$ is a t-distribution with $2\alpha_a$ degrees of freedom and it can be efficiently evaluated using standard statistical packages. It is proved that this procedure selects a with probability given by Equation 1. The major weakness of this approach is that it does not consider the improvement over the current policy that could arise from the choice of action a, i.e., it does not consider the *information gain* of exploration.

Myopic-VPI selection. The benefit of exploration can be estimated using the notion of *Value of Information* (VPI), i.e. the expected improvement in future decision quality that might arise from the information gathered during exploration. In short, this selection approach consists of a myopic approximation of the value of information for each action and leads to select the action that best balances exploration and exploitation. Roughly speaking, we need to balance the expected gains from exploration against the expected costs of doing a potentially suboptimal action.

The information gain $Gain_a(\mu_a^*)$, we get by learning the true value μ_a^* of μ_a can change the system policy only in two cases: (a) when it proves that an action previously considered sub-optimal is actually the best choice and (b) when it shows that an action previously considered best is actually worse than other actions. Since the value of μ_a^* is not known in advance, it is necessary to compute the *expected gain* given the system prior beliefs. Thus, the value of perfect information for action a will be:

$$VPI(a) = \int_{-\infty}^{+\infty} Gain_a(x) p(\mu_a = x) dx. \qquad (3)$$

This formula can be reduced to a closed form equation. Assume that a_1 is the action with the highest expected value and a_2 is the second best action; then $VPI(a)$ is equal to $c + (E[\mu_{a_2}] - E[\mu_{a_1}])p(\mu_{a_1} < E[\mu_{a_2}])$ if $a = a_1$, and it is equal to $c + (E[\mu_a] - E[\mu_{a_1}])p(\mu_a > E[\mu_{a_1}])$ otherwise. The procedure for computing c and a full explanation of this method are given in [4].

The cost of doing a potentially sub-optimal action is given by the difference between the value of a and the value of the current best action, i.e. $\max_{a'} \{E[\mu_{a'}] - E[\mu_a]\}$.

Finally, the value of action a can be computed as:

$$VPI(a) - (\max_{a'} \{E[\mu_{a'}] - E[\mu_a]\}), \qquad (4)$$

which is equivalent to $VPI(a) + E[\mu_a]$. The Myopic-VPI selection strategy selects the action that maximizes this value.

4.3 Posterior distributions updating

Let's now consider how to update the classifier distributions over predicted payoff after executing an action. Since posterior updating takes place in the current action set, in the following we omit the subscript 'a'. This consideration holds also for Section 4.4.

We can adopt an adaptation of *Moment Updating* technique based on the results of Theorem 1 [4]. Actually, we define distributions over *expected*, *total* rewards, but the available observations are instances of *actual*, *local* rewards; therefore, we cannot use Theorem 1 directly. Now, suppose that the system is in state s, executes action a, receives reward r, and ends up in state s'. Let R and $R_{s'}$ be two random variables denoting the expected total discounted sum of rewards received respectively from s and s' onwards.

The basic idea is to randomly sample values $R_{s'}^1, \cdots, R_{s'}^n$ from classifiers distribution, and then update the posterior $p(\mu, \tau)$ with the samples $r + \gamma R_{s'}^1, \cdots, r + \gamma R_{s'}^n$. Theorem 1 implies that we only need the first two moments of this sample to update our distribution; assuming that n tends to infinity, these two moments are:

$$\begin{aligned}M_1 &= E[r + \gamma R_{s'}] = r + \gamma E[R_{s'}], \qquad (5)\\ M_2 &= E\left[(r + \gamma R_{s'})^2\right] = E\left[r^2 + 2\gamma r R_{s'} + \gamma^2 R_{s'}^2\right]\\ &= r^2 + 2\gamma r E[R_{s'}] + \gamma^2 E\left[R_{s'}^2\right]. \qquad (6)\end{aligned}$$

If we assume that from s' onwards the system follows the current optimal policy, then $R_{s'}$ is a normally distributed variable. Thus, we can use the properties of the Normal-Gamma distribution to compute the expected values:

$$E[R_{s'}] = \mu_0, \quad (7)$$

$$E\left[R^2_{s'}\right] = \frac{\lambda+1}{\lambda}\frac{\beta}{\alpha-1} + \mu_0^2. \quad (8)$$

This approach results in a simple closed-form equation for updating classifiers hyperparameters. Unfortunately, it might become too confident of the value of the mean of μ[§]. In other words, uncertainty about $R_{s'}$ is not directly translated to uncertainty about the mean of R. To avoid this problem we can act on the β parameter by using some sort of discount factor η_β:

$$\beta_i' = (1-\eta_\beta)\beta_i + \eta_\beta\left(\frac{1}{2}(M_2 - M_1^2) + \frac{\lambda_i(M_1-\mu_{0_i})^2}{2(\lambda_i+1)}\right).$$

All the classifiers belonging to the same action set $[A]$ are updated with the same moments.

4.4 Fitness and genetics

Bayesian Q-Learning concepts lead straightforwardly to the definition of a *Bayesian accuracy* k_B based on the *precision* τ of classifiers instead of XCS error ε. The analytical expression of k_B is:

$$k_B = \begin{cases} 1 & \text{if } \beta < \beta_0, \\ \omega\left(\frac{\beta}{\beta_0}\right)^{-\nu} & \text{otherwise.} \end{cases}$$

Here β_0 controls indirectly the tolerance of the precision τ, and ω is a simple discount rate.

This fitness recalls the idea of a strong selection which is behind Wilson's accuracy [9]. It is obtained by (i) a threshold β_0 which distinguishes "good" classifiers from the "bad" ones, and (ii) a high rate of decline in goodness of classifiers, given by ω and ν parameters. As Wilson's accuracy rewards classifiers whose prediction error is below ε_0, our fitness function acts rewarding classifiers whose *precision* is above the threshold $1/\beta_0$.

The main difference between our Bayesian accuracy (based on precision) and Wilson's accuracy (based on prediction error) is that ours has a higher informative content than Wilson's. This is motivated by the following reasons: Bayesian accuracy is defined on the parameter β which (a) is inversely proportional to the mean and variance of the random variable τ, (b) is directly proportional to the variance of the random variable μ (Section 4.1), and (c) takes part in the updating process of the mean of μ (Theorem 1). In other words, Bayesian accuracy takes into account the uncertainty about the unknown payoffs through the β parameter, which affects the distributions over both the mean and the precision of the unknown related random variable R. Roughly speaking, in a probabilistic context Wilson's accuracy can be considered as a surrogate just for the variance of μ.

In our way of thinking, a Bayesian fitness function can take into account the uncertainty on the effects of agent's actions in a stochastic environment better than a non probabilistic one. To have a fair comparison with Wilson's accuracy, we can determine the threshold β_0 from ε_0 by the following relationship: $P(r-\varepsilon_0 < \mu < r+\varepsilon_0) = 0.999$. We can assume that $\mu \sim N(r, \beta_0)$, and so compute the value of β_0 from the standard normal distribution table: $\beta_0 = (\frac{\varepsilon_0}{3.2905})^2$.

[§]Note that the parameter λ can be intended as the confidence in the estimate of the unknown mean of μ.

5. EXPERIMENTAL VALIDATION

The experiments we present in this paper have been conducted in the well-known "multiplexer" problems and "woods" environments, which are common benchmarks in the XCS community. All experiments in this paper were carried out with Lanzi's `xcslib` [6]. The description of the tasks and input codings will be here summarized leaving the reader to [9] for a complete description. Any statistics presented in this paper is averaged over ten experiments. In each plot we use the performance of XCS with no generalization to indicate optimal performance.

5.1 The Boolean Multiplexer Problem

Boolean multiplexers are defined for strings of l bits, where $l = k + 2^k$; the first k bits $(x_0, \cdots, x_{k-1})$ represent an address which indexes the remaining 2^k bits $(y_0, \cdots, y_{2^k-1})$. The function returns the value of the indexed bit. For instance, in the 6-multiplexer function mp_6, we have that $mp_6(100010) = 1$. Though they are "one-step" (i.e. not sequential) artificial environments, we consider them because they make it possible to study generalization without the added complications that sequential environments entail. The performance of each system is computed as the average reward in the last 50 testing problems and plots are obtained averaging 10 runs of the algorithm.

20-Multiplexer. We begin by comparing results on the 20-multiplexer problem using the following parameters: $\beta = 0.2$, $\alpha = 0.1$, $\gamma = 0.7$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{ga} = 25$, $p_\chi = 0.8$, $p_\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, and $\eta_\beta = 0.8$. Population is set to 2000 classifiers and $P_\# = 0.6$ and XCS parameters are those reported in [9]. Figure 1 shows that BXCS reaches 100% performance a few problems before XCS, but the two systems converge to the same population size, i.e. they show the same generalization capabilities.

37-Multiplexer. In these simulations we used the same parameter settings of the previous experiments, except for population size, which is set to 5000 classifiers. Figure 2 shows that XCS reaches 100% performance before BXCS, but examination of the populations reveals that BXCS seems to have a higher tendency to generalize than XCS, i.e., it reaches a more compact final population.

5.2 The deterministic woods environment

Woods environments are grid worlds in which each cell can be empty, represented as ".", can contain an obstacle, represented by "T" (tree), or otherwise food, "F". An animat placed in the environment must learn to reach food cells. The animat senses the environment by eight sensors, one for each adjacent cell, and the perception is represented as a binary string of 16 digits (for environments with only one type of food and obstacles). The animat can decide to move in any of the adjacent cells. If the destination cell is free then the animat moves; if the cell contains food the animat moves and eats the food receiving a reward (e.g. 1000), while if the destination cell contains a tree the move does not take place. In these environments, the agent has probability 1 of reaching the correct destination.

Each experiment focuses on a number of problems that the animat must solve. For each problem the animat is randomly placed in a blank cell of the environment; it then moves under the control of the system until it enters a food

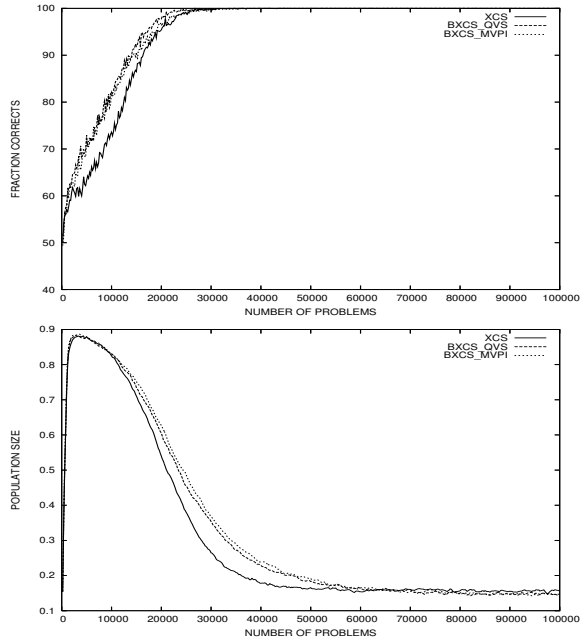

Figure 1: 20-multiplexer with 1000/0 reward

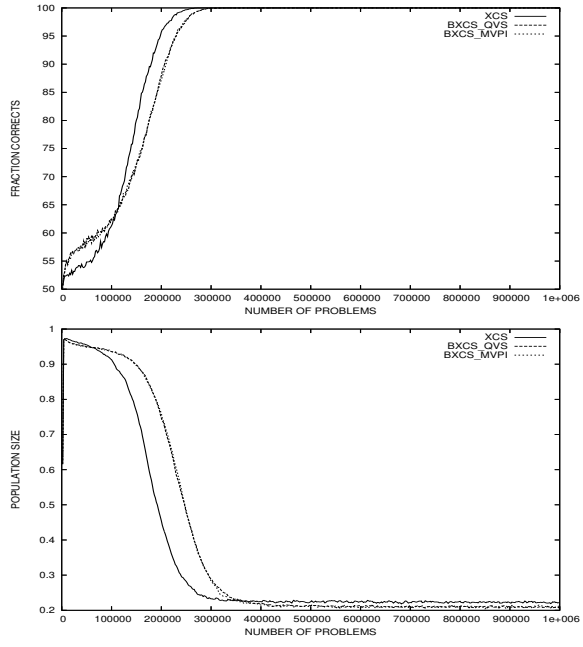

Figure 2: 37-multiplexer with 1000/0 reward

cell, eats the food and receives the reward. The food immediately re-grows and a new problem begins. The performance of each system is computed as the average number of steps to a goal position in the last 50 testing problems and plots are obtained averaging 10 runs of the algorithm.

Woods1. We tested the systems with the following parameters: $\beta = 0.2$, $\alpha = 0.1$, $\gamma = 0.7$, $\varepsilon_0 = 0.01$, $\nu = 5$, $\theta_{ga} = 25$, $p_\chi = 0.8$, $p_\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\eta_\beta = 0.2$. Population is set to 400 classifiers and $P_\# = 0.5$. As we

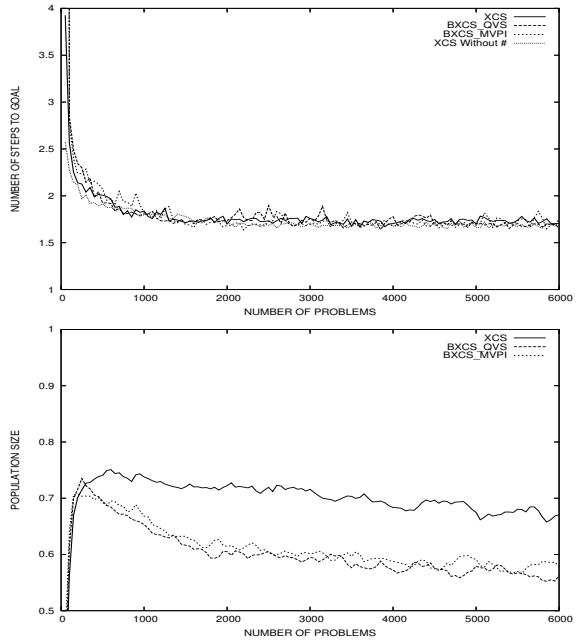

Figure 3: Deterministic Woods1 with semiuniform action selection probability set to 0.25

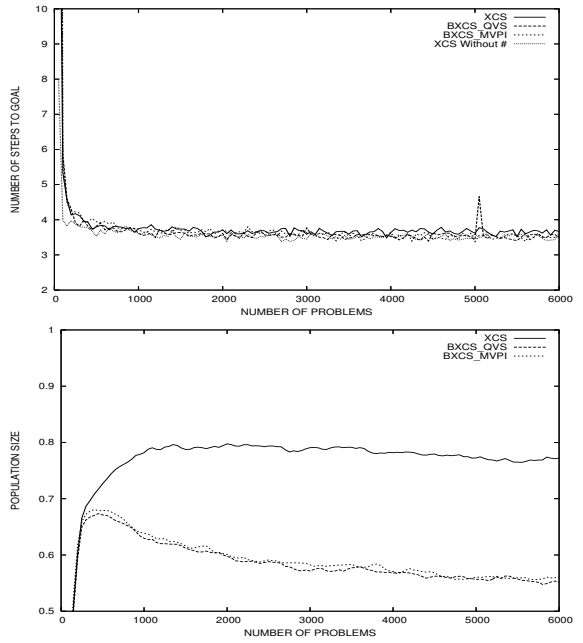

Figure 4: Deterministic Maze4 with semiuniform action selection probability set to 0.25

can see from Figure 3, when semiuniform action selection probability is set to 0.25, i.e., in the exploration phase, the best action is selected with probability $p = 0.75$, the two systems show the analogous performance (Figure 3) while BXCS seems to have a higher generalization capability.

Maze4. In these simulations we use the same parameter settings used in experiments on Woods1, except for popula-

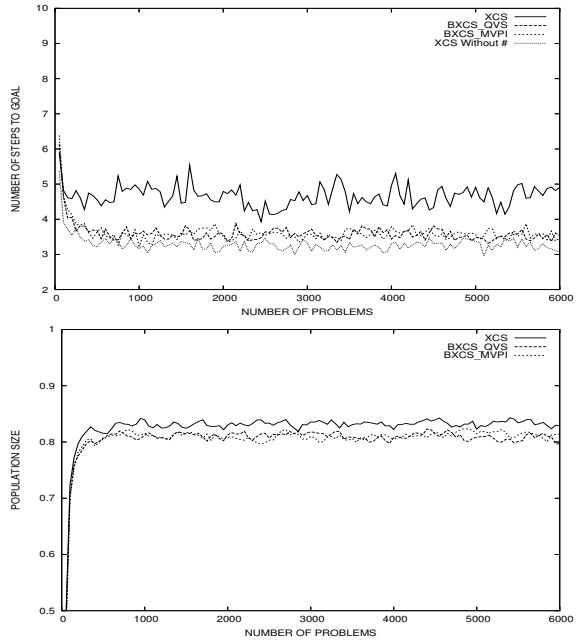

Figure 5: Stochastic Woods1 with $\epsilon = 0.5$ and semiuniform action selection probability set to 0.25

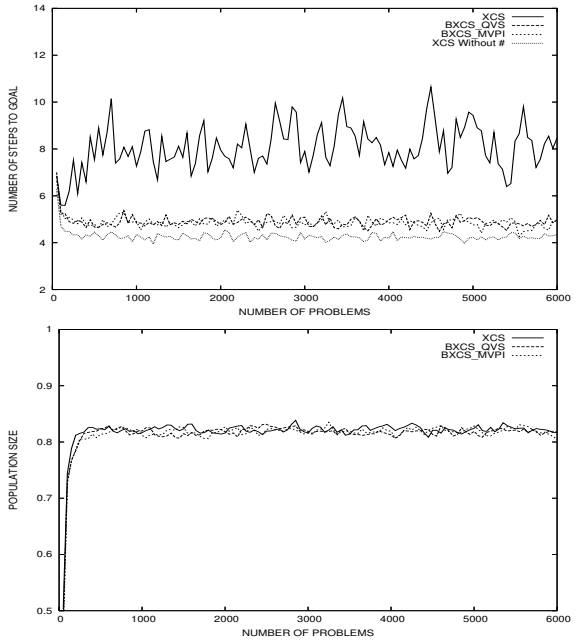

Figure 6: Stochastic Woods1 with $\epsilon = 0.66$ and semiuniform action selection probability set to 0.25

tion which is set to 1600 classifiers. Experiments reported in Figure 4 show that with a uniform action selection probability set to 0.25, XCS seems to reach an optimal solution, but it has lower generalization capabilities than BXCS.

5.3 Stochastic woods environment

In the stochastic version of woods environment, when the agent tries to move in a certain direction it has probability $1 - \epsilon$ ($0 \leq \epsilon < 1$) of reaching the correct destination; it has probability ϵ of slipping, i.e., reaching one of the two positions next to the original destination on the left or on the right. The value ϵ represents the *degree of uncertainty* affecting agent actions.

Woods1. A first analysis of the agent performance in stochastic environments can be drawn using the simple Woods1 task. General parameters are as follows: $\beta = 0.2$, $\alpha = 0.1$, $\gamma = 0.7$, $\varepsilon_0 = 0.01$, $\nu = 5$, $\theta_{ga} = 25$, $p_\chi = 0.8$, $p_\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\eta_\beta = 0.2$. Population size is set to 400 classifiers and $P_\# = 0.5$. Figure 5 shows the results obtained when $\epsilon = 0.5$. XCS converges to a sub-optimal solution, whereas BXCS comes close to the optimum. Note that BXCS cannot stably reach optimal performance because it has to deal with overgeneralization. A higher level of uncertainty, i.e. $\epsilon = 0.66$, makes XCS unable to learn from its experience (see Figure 6). BXCS can cope with this new worst environmental situation, converging near to the optimum, without losing its generalization capabilities. We'll go deeper into this question by analyzing the results coming from stochastic Maze4 task.

Maze4. For stochastic maze4 environment we ran experiments for increasing values of sliding probability (ϵ). General parameters are as follows: $\beta = 0.2$, $\alpha = 0.1$, $\gamma = 0.7$, $\varepsilon_0 = 0.01$, $\nu = 5$, $\theta_{ga} = 25$, $p_\chi = 0.8$, $p_\mu = 0.04$, $\theta_{del} = 20$, and $\delta = 0.1$. Population is set to 1600 classifiers and $P_\# = 0.5$. Finally, we set semiuniform random action selection probability to 0.25. Figure 7 shows that when the uncertainty on the agent actions is less than or equal to 0.5, XCS tends to converge close to optimal performance, while BXCS becomes stable very close to the optimum after a few number of problems. As noted in stochastic woods1 environment, BXCS cannot reach optimal performance because it

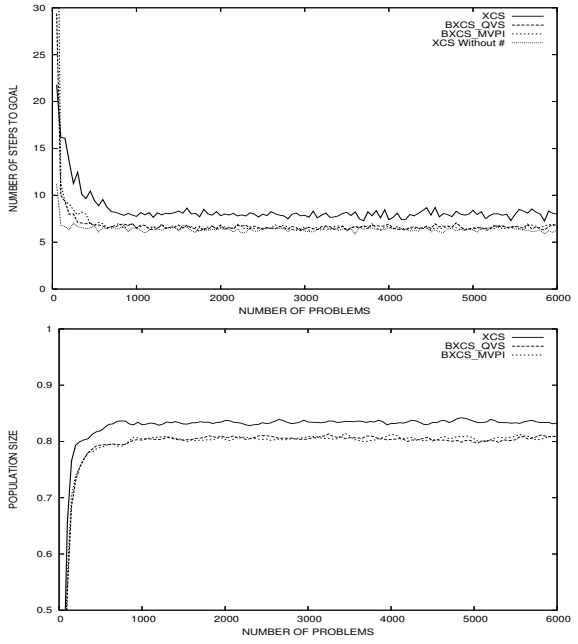

Figure 7: Stochastic Maze4 with $\epsilon = 0.5$ and semiuniform action selection probability set to 0.25

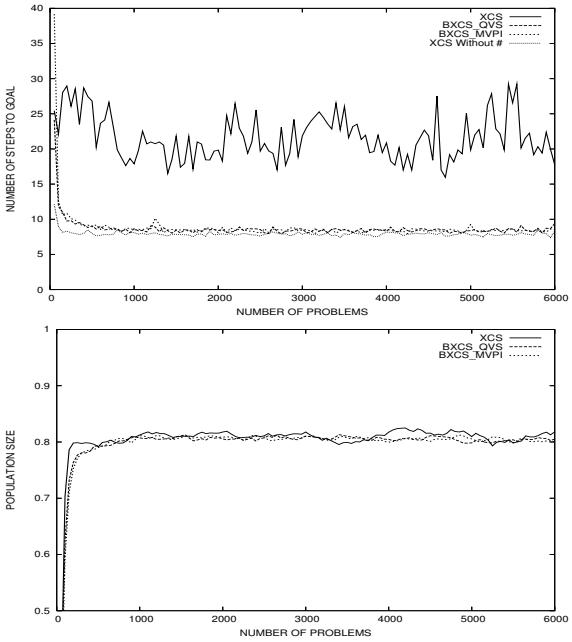

Figure 8: Stochastic Maze4 with $\epsilon = 0.66$ and semi-uniform action selection probability set to 0.25

has to deal with overgeneralization. However, when uncertainty is higher, i.e. ϵ is 0.66, the previous considerations do not hold anymore; it is important to specify that for $\epsilon = 0.66$ the effects of agent actions are almost unpredictable. The results plotted in Figure 8 give evidence to the drastic drop of performance of XCS, which rapidly becomes completely unstable. On the other hand, BXCS can cope with these levels of uncertainty and converge close to the optimal solution. This happens because BXCS is able to distinguish between the loss of precision (τ, see Section 4) due to overgeneralization and the one due to uncertainty of the environment. On the basis of this valuable skill we mark out the parameter β (Section 4) and its updating mechanism. We update classifier posteriors and select actions by means of joint distributions defined on environmental niches; as a result, all the classifiers belonging to the same action set (niche) feel, share, and suffer the same degree of uncertainty about the environment. From this point of view, BXCS appears to be a useful tool for incorporating various forms of uncertainty into a uniform framework.

6. CONCLUSION

In this paper, we have introduced a new approach to XCS, based on a Bayesian framework. The definition of probability distributions over classifiers payoff estimates induces a novel interpretation of classifier founded on the concept of *precision*. We extended the notion of accuracy exploiting the *value of information* gained during exploration. We showed that our system can cope with a high level of uncertainty, guaranteeing the convergence to the optimum and considerable generalization capabilities as summarized by the results reported in Table 6.

As a matter of fact, the strategy of maintaining and propagating probability distribution over the classifier estimate of payoff reflects uncertainty: the decisions are nearly deterministic when the distributions are sharp, and conversely, nearly random when they are flat. XCS is vulnerable to noise, i.e., when the degree of uncertainty is high its performance may break down drastically [7]; BXCS shows the ability both to reason despite uncertainty and to decide taking uncertainty into account.

ϵ	XCS	BXCS	XCS without #
0	1.75	1.75	1.75
0.5	4.7	3.5	3.25
0.66	8	4.95	4.25

ϵ	XCS	BXCS	XCS without #
0	3.7	3.6	3.5
0.5	8	6.6	6.5
0.66	20	8.3	7.8

Table 1: Average number of steps to goal in Woods1 (above) and Maze4 (below)

We are currently investigating the opportunity to use the Bayesian approach to define a new fitness function. The main idea is that Bayes theorem can be used to estimate the posterior fitness of classifiers from their prior fitness values, thus implementing a form of "Bayesian evolution".

7. ADDITIONAL AUTHORS

Additional authors: Andrea Bonarini (Politecnico di Milano, Department of Electronics and Information, via Ponzio 34/5, I-20133, Milan, Italy, email: `bonarini@elet.polimi.it`)

8. REFERENCES

[1] M. Butz, T. Kovacs, P. Lanzi, and S. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Trans. On Evolutionary Computation*, 8(1), 2004.

[2] M. Butz and S. Wilson. An algorithmic description of XCS. In P. Lanzi, W. Stolzmann, and S. Wilson, editors, *Advances in Learning Classifier Systems: Proc. of the 3rd Int. Workshop*, pages 253–272. Berlin, Germany: Springer-Verlag, 2001.

[3] G. Casella and R. Berger. *Statistical Inference*. Wadsworth & Brooks/Cole, 1990.

[4] R. Dearden, N. Friedman, and S. Russel. Bayesian q-learning. In *Proc. of the 15th National Conference on Artificial Intelligence*, Menlo Park, CA, 1998.

[5] P. Lanzi. An analysis of the memory mechanism of XCS. In *Proc. of the 3rd Annu. Genetic Programming Conference*, pages 593–623. Morgan Kaufmann, 1998.

[6] P. Lanzi. *The XCS Library*. 2003. http://xcslib.sourceforge.net.

[7] P. Lanzi and M. Colombetti. An extension to the XCS classifier systems for stochastic environments. In *Proc. of the Genetic and Evolutionary Computation Conference*, pages 345–352. Morgan Kauffman:San Francisco, CA, 1999.

[8] R. Sutton and A. Barto. *Reinforcement Learning: An introduction*. Cambridge, MA: Mit Press, 1998.

[9] S. Wilson. Classifier fitness based on accuracy. *Evol. Comput.*, 3(2):149–175, 1995.