Journal of Artificial Intelligence Research 40 (2011) 95-142 Submitted 07/10; published 01/11

A Monte-Carlo AIXI Approximation

Joel Veness JOELV@CSE.UNSW.EDU.AU
University of New South Wales and National ICT Australia

Kee Siong Ng KEESIONG.NG@GMAIL.COM
The Australian National University

Marcus Hutter MARCUS.HUTTER@ANU.EDU.AU
The Australian National University and National ICT Australia

William Uther WILLIAM.UTHER@NICTA.COM.AU
National ICT Australia and University of New South Wales

David Silver DAVIDSTARSILVER@GOOGLEMAIL.COM
Massachusetts Institute of Technology

Abstract

This paper introduces a principled approach for the design of a scalable general reinforcement learning agent. Our approach is based on a direct approximation of AIXI, a Bayesian optimality notion for general reinforcement learning agents. Previously, it has been unclear whether the theory of AIXI could motivate the design of practical algorithms. We answer this hitherto open question in the affirmative, by providing the first computationally feasible approximation to the AIXI agent. To develop our approximation, we introduce a new Monte-Carlo Tree Search algorithm along with an agent-specific extension to the Context Tree Weighting algorithm. Empirically, we present a set of encouraging results on a variety of stochastic and partially observable domains. We conclude by proposing a number of directions for future research.

1. Introduction

Reinforcement Learning (Sutton & Barto, 1998) is a popular and influential paradigm for agents that learn from experience. AIXI (Hutter, 2005) is a Bayesian optimality notion for reinforcement learning agents in unknown environments. This paper introduces and evaluates a practical reinforcement learning agent that is directly inspired by the AIXI theory.

1.1 The General Reinforcement Learning Problem

Consider an agent that exists within some unknown environment. The agent interacts with the environment in cycles. In each cycle, the agent executes an action and in turn receives an observation and a reward. The only information available to the agent is its history of previous interactions. The *general reinforcement learning problem* is to construct an agent that, over time, collects as much reward as possible from the (unknown) environment.

1.2 The AIXI Agent

The AIXI agent is a mathematical solution to the general reinforcement learning problem. To achieve generality, the environment is assumed to be an unknown but computable function; i.e.

the observations and rewards received by the agent, given its past actions, can be computed by some program running on a Turing machine. The AIXI agent results from a synthesis of two ideas:

1. the use of a finite-horizon expectimax operation from sequential decision theory for action selection; and
2. an extension of Solomonoff's universal induction scheme (Solomonoff, 1964) for future prediction in the agent context.

More formally, let $U(q, a_1a_2 \ldots a_n)$ denote the output of a universal Turing machine U supplied with program q and input $a_1a_2 \ldots a_n$, $m \in \mathbb{N}$ a finite lookahead horizon, and $\ell(q)$ the length in bits of program q. The action picked by AIXI at time t, having executed actions $a_1a_2 \ldots a_{t-1}$ and having received the sequence of observation-reward pairs $o_1r_1o_2r_2 \ldots o_{t-1}r_{t-1}$ from the environment, is given by:

$$a_t^* = \arg\max_{a_t} \sum_{o_tr_t} \ldots \max_{a_{t+m}} \sum_{o_{t+m}r_{t+m}} [r_t + \cdots + r_{t+m}] \sum_{q:U(q,a_1\ldots a_{t+m})=o_1r_1\ldots o_{t+m}r_{t+m}} 2^{-\ell(q)}. \tag{1}$$

Intuitively, the agent considers the sum of the total reward over all possible futures up to m steps ahead, weighs each of them by the complexity of programs consistent with the agent's past that can generate that future, and then picks the action that maximises expected future rewards. Equation (1) embodies in one line the major ideas of Bayes, Ockham, Epicurus, Turing, von Neumann, Bellman, Kolmogorov, and Solomonoff. The AIXI agent is rigorously shown by Hutter (2005) to be optimal in many different senses of the word. In particular, the AIXI agent will rapidly learn an accurate model of the environment and proceed to act optimally to achieve its goal.

Accessible overviews of the AIXI agent have been given by both Legg (2008) and Hutter (2007). A complete description of the agent can be found in the work of Hutter (2005).

1.3 AIXI as a Principle

As the AIXI agent is only asymptotically computable, it is by no means an algorithmic solution to the general reinforcement learning problem. Rather it is best understood as a Bayesian *optimality notion* for decision making in general unknown environments. As such, its role in general AI research should be viewed in, for example, the same way the minimax and empirical risk minimisation principles are viewed in decision theory and statistical machine learning research. These principles define what is optimal behaviour if computational complexity is not an issue, and can provide important theoretical guidance in the design of practical algorithms. This paper demonstrates, for the first time, how a practical agent can be built from the AIXI theory.

1.4 Approximating AIXI

As can be seen in Equation (1), there are two parts to AIXI. The first is the expectimax search into the future which we will call *planning*. The second is the use of a Bayesian mixture over Turing machines to predict future observations and rewards based on past experience; we will call that *learning*. Both parts need to be approximated for computational tractability. There are many different approaches one can try. In this paper, we opted to use a generalised version of the UCT algorithm (Kocsis & Szepesvári, 2006) for planning and a generalised version of the Context Tree Weighting algorithm (Willems, Shtarkov, & Tjalkens, 1995) for learning. This combination of ideas, together with the attendant theoretical and experimental results, form the main contribution of this paper.

1.5 Paper Organisation

The paper is organised as follows. Section 2 introduces the notation and definitions we use to describe environments and accumulated agent experience, including the familiar notions of reward, policy and value functions for our setting. Section 3 describes a general Bayesian approach for learning a model of the environment. Section 4 then presents a Monte-Carlo Tree Search procedure that we will use to approximate the expectimax operation in AIXI. This is followed by a description of the Context Tree Weighting algorithm and how it can be generalised for use in the agent setting in Section 5. We put the two ideas together in Section 6 to form our AIXI approximation algorithm. Experimental results are then presented in Sections 7. Section 8 provides a discussion of related work and the limitations of our current approach. Section 9 highlights a number of areas for future investigation.

2. The Agent Setting

This section introduces the notation and terminology we will use to describe strings of agent experience, the true underlying environment and the agent's model of the true environment.

Notation. A string $x_1x_2\dots x_n$ of length n is denoted by $x_{1:n}$. The prefix $x_{1:j}$ of $x_{1:n}$, $j \leq n$, is denoted by $x_{\leq j}$ or $x_{<j+1}$. The notation generalises for blocks of symbols: e.g. $ax_{1:n}$ denotes $a_1x_1a_2x_2\dots a_nx_n$ and $ax_{<j}$ denotes $a_1x_1a_2x_2\dots a_{j-1}x_{j-1}$. The empty string is denoted by ϵ. The concatenation of two strings s and r is denoted by sr.

2.1 Agent Setting

The (finite) action, observation, and reward spaces are denoted by $\mathcal{A}, \mathcal{O}$, and $\mathcal{R}$ respectively. Also, $\mathcal{X}$ denotes the joint perception space $\mathcal{O} \times \mathcal{R}$.

Definition 1. *A history h is an element of* $(\mathcal{A} \times \mathcal{X})^* \cup (\mathcal{A} \times \mathcal{X})^* \times \mathcal{A}$.

The following definition states that the environment takes the form of a probability distribution over possible observation-reward sequences conditioned on actions taken by the agent.

Definition 2. *An environment ρ is a sequence of conditional probability functions* $\{\rho_0, \rho_1, \rho_2, \dots\}$, *where $\rho_n \colon \mathcal{A}^n \to$ Density $(\mathcal{X}^n)$, that satisfies*

$$\forall a_{1:n} \forall x_{<n} : \rho_{n-1}(x_{<n} \,|\, a_{<n}) = \sum_{x_n \in \mathcal{X}} \rho_n(x_{1:n} \,|\, a_{1:n}). \tag{2}$$

In the base case, we have $\rho_0(\epsilon \,|\, \epsilon) = 1$.

Equation (2), called the chronological condition in (Hutter, 2005), captures the natural constraint that action a_n has no effect on earlier perceptions $x_{<n}$. For convenience, we drop the index n in ρ_n from here onwards.

Given an environment ρ, we define the predictive probability

$$\rho(x_n \,|\, ax_{<n}a_n) := \frac{\rho(x_{1:n} \,|\, a_{1:n})}{\rho(x_{<n} \,|\, a_{<n})} \tag{3}$$

$\forall a_{1:n} \forall x_{1:n}$ such that $\rho(x_{<n} \,|\, a_{<n}) > 0$. It now follows that

$$\rho(x_{1:n} \,|\, a_{1:n}) = \rho(x_1 \,|\, a_1)\rho(x_2 \,|\, ax_1a_2) \cdots \rho(x_n \,|\, ax_{<n}a_n). \tag{4}$$

Definition 2 is used in two distinct ways. The first is a means of describing the true underlying environment. This may be unknown to the agent. Alternatively, we can use Definition 2 to describe an agent's *subjective* model of the environment. This model is typically learnt, and will often only be an approximation to the true environment. To make the distinction clear, we will refer to an agent's *environment model* when talking about the agent's model of the environment.

Notice that $\rho(\cdot \mid h)$ can be an arbitrary function of the agent's previous history h. Our definition of environment is sufficiently general to encapsulate a wide variety of environments, including standard reinforcement learning setups such as MDPs or POMDPs.

2.2 Reward, Policy and Value Functions

We now cast the familiar notions of *reward*, *policy* and *value* (Sutton & Barto, 1998) into our setup. The agent's goal is to accumulate as much reward as it can during its lifetime. More precisely, the agent seeks a *policy* that will allow it to maximise its expected future reward up to a fixed, finite, but arbitrarily large horizon $m \in \mathbb{N}$. The instantaneous reward values are assumed to be bounded. Formally, a policy is a function that maps a history to an action. If we define $R_k(aor_{\leq t}) := r_k$ for $1 \leq k \leq t$, then we have the following definition for the expected future value of an agent acting under a particular policy:

Definition 3. *Given history $ax_{1:t}$, the m-horizon expected future reward of an agent acting under policy $\pi\colon (\mathcal{A} \times \mathcal{X})^* \to \mathcal{A}$ with respect to an environment ρ is:*

$$v_\rho^m(\pi, ax_{1:t}) := \mathbb{E}_\rho \left[\sum_{i=t+1}^{t+m} R_i(ax_{\leq t+m}) \,\middle|\, x_{1:t} \right], \tag{5}$$

where for $t < k \leq t + m$, $a_k := \pi(ax_{<k})$. The quantity $v_\rho^m(\pi, ax_{1:t}a_{t+1})$ is defined similarly, except that a_{t+1} is now no longer defined by π.

The optimal policy π^* is the policy that maximises the expected future reward. The maximal achievable expected future reward of an agent with history h in environment ρ looking m steps ahead is $V_\rho^m(h) := v_\rho^m(\pi^*, h)$. It is easy to see that if $h \in (\mathcal{A} \times \mathcal{X})^t$, then

$$V_\rho^m(h) = \max_{a_{t+1}} \sum_{x_{t+1}} \rho(x_{t+1} \mid ha_{t+1}) \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \rho(x_{t+m} \mid hax_{t+1:t+m-1}a_{t+m}) \left[\sum_{i=t+1}^{t+m} r_i \right]. \tag{6}$$

For convenience, we will often refer to Equation (6) as the *expectimax operation*. Furthermore, the m-horizon optimal action a^*_{t+1} at time $t + 1$ is related to the expectimax operation by

$$a^*_{t+1} = \arg\max_{a_{t+1}} V_\rho^m(ax_{1:t}a_{t+1}). \tag{7}$$

Equations (5) and (6) can be modified to handle discounted reward, however we focus on the finite-horizon case since it both aligns with AIXI and allows for a simplified presentation.

3. Bayesian Agents

As mentioned earlier, Definition 2 can be used to describe the agent's subjective model of the true environment. Since we are assuming that the agent does not initially know the true environment, we desire subjective models whose predictive performance improves as the agent gains experience. One way to provide such a model is to take a Bayesian perspective. Instead of committing to any single fixed environment model, the agent uses a *mixture* of environment models. This requires committing to a class of possible environments (the model class), assigning an initial weight to each possible environment (the prior), and subsequently updating the weight for each model using Bayes rule (computing the posterior) whenever more experience is obtained. The process of learning is thus implicit within a Bayesian setup.

The mechanics of this procedure are reminiscent of Bayesian methods to predict sequences of (single typed) observations. The key difference in the agent setup is that each prediction may now also depend on previous agent actions. We incorporate this by using the *action conditional* definitions and identities of Section 2.

Definition 4. *Given a countable model class* $\mathcal{M} := \{\rho_1, \rho_2, \dots\}$ *and a prior weight* $w_0^\rho > 0$ *for each* $\rho \in \mathcal{M}$ *such that* $\sum_{\rho\in\mathcal{M}} w_0^\rho = 1$*, the mixture environment model is* $\xi(x_{1:n} \,|\, a_{1:n}) := \sum\limits_{\rho\in\mathcal{M}} w_0^\rho \rho(x_{1:n} \,|\, a_{1:n})$.

The next proposition allows us to use a mixture environment model whenever we can use an environment model.

Proposition 1. *A mixture environment model is an environment model.*

Proof. $\forall a_{1:n} \in \mathcal{A}^n$ and $\forall x_{<n} \in \mathcal{X}^{n-1}$ we have that

$$\sum_{x_n\in\mathcal{X}} \xi(x_{1:n} \,|\, a_{1:n}) = \sum_{x_n\in\mathcal{X}} \sum_{\rho\in\mathcal{M}} w_0^\rho \rho(x_{1:n} \,|\, a_{1:n}) = \sum_{\rho\in\mathcal{M}} w_0^\rho \sum_{x_n\in\mathcal{X}} \rho(x_{1:n} \,|\, a_{1:n}) = \xi(x_{<n} \,|\, a_{<n})$$

where the final step follows from application of Equation (2) and Definition 4. □

The importance of Proposition 1 will become clear in the context of planning with environment models, described in Section 4.

3.1 Prediction with a Mixture Environment Model

As a mixture environment model is an environment model, we can simply use:

$$\xi(x_n \,|\, ax_{<n}a_n) = \frac{\xi(x_{1:n} \,|\, a_{1:n})}{\xi(x_{<n} \,|\, a_{<n})} \tag{8}$$

to predict the next observation reward pair. Equation (8) can also be expressed in terms of a convex combination of model predictions, with each model weighted by its posterior, from

$$\xi(x_n \,|\, ax_{<n}a_n) = \frac{\sum\limits_{\rho\in\mathcal{M}} w_0^\rho \rho(x_{1:n} \,|\, a_{1:n})}{\sum\limits_{\rho\in\mathcal{M}} w_0^\rho \rho(x_{<n} \,|\, a_{<n})} = \sum_{\rho\in\mathcal{M}} w_{n-1}^\rho \rho(x_n \,|\, ax_{<n}a_n),$$

where the posterior weight w_{n-1}^ρ for environment model ρ is given by

$$w_{n-1}^\rho := \frac{w_0^\rho \rho(x_{<n} \,|\, a_{<n})}{\sum\limits_{\nu\in\mathcal{M}} w_0^\nu \nu(x_{<n} \,|\, a_{<n})} = \Pr(\rho \,|\, ax_{<n}) \tag{9}$$

If $|\mathcal{M}|$ is finite, Equations (8) and (3.1) can be maintained online in $O(|\mathcal{M}|)$ time by using the fact that

$$\rho(x_{1:n} \,|\, a_{1:n}) = \rho(x_{<n} \,|\, a_{<n})\rho(x_n \,|\, ax_{<n}a),$$

which follows from Equation (4), to incrementally maintain the likelihood term for each model.

3.2 Theoretical Properties

We now show that if there is a good model of the (unknown) environment in $\mathcal{M}$, an agent using the mixture environment model

$$\xi(x_{1:n} \,|\, a_{1:n}) := \sum_{\rho \in \mathcal{M}} w_0^{\rho} \rho(x_{1:n} \,|\, a_{1:n}) \tag{10}$$

will predict well. Our proof is an adaptation from the work of Hutter (2005). We present the full proof here as it is both instructive and directly relevant to many different kinds of practical Bayesian agents.

First we state a useful entropy inequality.

Lemma 1 (Hutter, 2005). *Let $\{y_i\}$ and $\{z_i\}$ be two probability distributions, i.e. $y_i \geq 0, z_i \geq 0$, and $\sum_i y_i = \sum_i z_i = 1$. Then we have*

$$\sum_i (y_i - z_i)^2 \leq \sum_i y_i \ln \frac{y_i}{z_i}.$$

Theorem 1. *Let μ be the true environment. The μ-expected squared difference of μ and ξ is bounded as follows. For all $n \in \mathbb{N}$, for all $a_{1:n}$,*

$$\sum_{k=1}^{n} \sum_{x_{1:k}} \mu(x_{<k} \,|\, a_{<k}) \Big(\mu(x_k \,|\, ax_{<k}a_k) - \xi(x_k \,|\, ax_{<k}a_k) \Big)^2 \leq \min_{\rho \in \mathcal{M}} \Big\{ -\ln w_0^{\rho} + D_{1:n}(\mu \,\|\, \rho) \Big\},$$

where $D_{1:n}(\mu \,\|\, \rho) := \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\mu(x_{1:n} \,|\, a_{1:n})}{\rho(x_{1:n} \,|\, a_{1:n})}$ is the KL divergence of $\mu(\cdot \,|\, a_{1:n})$ and $\rho(\cdot \,|\, a_{1:n})$.

Proof. Combining Sections 3.2.8 and 5.1.3 from the work of Hutter (2005) we get

$$\begin{aligned}
&\sum_{k=1}^{n} \sum_{x_{1:k}} \mu(x_{<k} \,|\, a_{<k}) \Big(\mu(x_k \,|\, ax_{<k}a_k) - \xi(x_k \,|\, ax_{<k}a_k) \Big)^2 \\
&= \sum_{k=1}^{n} \sum_{x_{<k}} \mu(x_{<k} \,|\, a_{<k}) \sum_{x_k} \Big(\mu(x_k \,|\, ax_{<k}a_k) - \xi(x_k \,|\, ax_{<k}a_k) \Big)^2 \\
&\leq \sum_{k=1}^{n} \sum_{x_{<k}} \mu(x_{<k} \,|\, a_{<k}) \sum_{x_k} \mu(x_k \,|\, ax_{<k}a_k) \ln \frac{\mu(x_k \,|\, ax_{<k}a_k)}{\xi(x_k \,|\, ax_{<k}a_k)} && \text{[Lemma 1]} \\
&= \sum_{k=1}^{n} \sum_{x_{1:k}} \mu(x_{1:k} \,|\, a_{1:k}) \ln \frac{\mu(x_k \,|\, ax_{<k}a_k)}{\xi(x_k \,|\, ax_{<k}a_k)} && \text{[Equation (3)]} \\
&= \sum_{k=1}^{n} \sum_{x_{1:k}} \Big(\sum_{x_{k+1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \Big) \ln \frac{\mu(x_k \,|\, ax_{<k}a_k)}{\xi(x_k \,|\, ax_{<k}a_k)} && \text{[Equation (2)]} \\
&= \sum_{k=1}^{n} \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\mu(x_k \,|\, ax_{<k}a_k)}{\xi(x_k \,|\, ax_{<k}a_k)} \\
&= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \sum_{k=1}^{n} \ln \frac{\mu(x_k \,|\, ax_{<k}a_k)}{\xi(x_k \,|\, ax_{<k}a_k)} \\
&= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\mu(x_{1:n} \,|\, a_{1:n})}{\xi(x_{1:n} \,|\, a_{1:n})} && \text{[Equation (4)]}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \left[\frac{\mu(x_{1:n} \,|\, a_{1:n})}{\rho(x_{1:n} \,|\, a_{1:n})} \frac{\rho(x_{1:n} \,|\, a_{1:n})}{\xi(x_{1:n} \,|\, a_{1:n})} \right] && [\text{arbitrary } \rho \in \mathcal{M}] \\
&= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\mu(x_{1:n} \,|\, a_{1:n})}{\rho(x_{1:n} \,|\, a_{1:n})} + \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\rho(x_{1:n} \,|\, a_{1:n})}{\xi(x_{1:n} \,|\, a_{1:n})} \\
&\leq D_{1:n}(\mu \,\|\, \rho) + \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\rho(x_{1:n} \,|\, a_{1:n})}{w_0^\rho \rho(x_{1:n} \,|\, a_{1:n})} && [\text{Definition 4}] \\
&= D_{1:n}(\mu \,\|\, \rho) - \ln w_0^\rho.
\end{aligned}
$$

Since the inequality holds for arbitrary $\rho \in \mathcal{M}$, it holds for the minimising ρ. □

In Theorem 1, take the supremum over n in the r.h.s and then the limit $n \to \infty$ on the l.h.s. If $\sup_n D_{1:n}(\mu \,\|\, \rho) < \infty$ for the minimising ρ, the infinite sum on the l.h.s can only be finite if $\xi(x_k \,|\, ax_{<k}a_k)$ converges sufficiently fast to $\mu(x_k \,|\, ax_{<k}a_k)$ for $k \to \infty$ with probability 1, hence ξ predicts μ with rapid convergence. As long as $D_{1:n}(\mu \,\|\, \rho) = o(n)$, ξ still converges to μ but in a weaker Cesàro sense. The contrapositive of the statement tells us that if ξ fails to predict the environment well, then there is no good model in $\mathcal{M}$.

3.3 AIXI: The Universal Bayesian Agent

Theorem 1 motivates the construction of Bayesian agents that use rich model classes. The AIXI agent can be seen as the limiting case of this viewpoint, by using the largest model class expressible on a Turing machine.

Note that AIXI can handle stochastic environments since Equation (1) can be shown to be formally equivalent to

$$
a_t^* = \arg\max_{a_t} \sum_{o_t r_t} \ldots \max_{a_{t+m}} \sum_{o_{t+m} r_{t+m}} [r_t + \cdots + r_{t+m}] \sum_{\rho \in \mathcal{M}_U} 2^{-K(\rho)} \rho(x_{1:t+m} \,|\, a_{1:t+m}), \tag{11}
$$

where $\rho(x_{1:t+m} \,|\, a_1 \ldots a_{t+m})$ is the probability of observing $x_1 x_2 \ldots x_{t+m}$ given actions $a_1 a_2 \ldots a_{t+m}$, class $\mathcal{M}_U$ consists of all enumerable chronological semimeasures (Hutter, 2005), which includes all computable ρ, and $K(\rho)$ denotes the Kolmogorov complexity (Li & Vitányi, 2008) of ρ with respect to U. In the case where the environment is a computable function and

$$
\xi_U(x_{1:t} \,|\, a_{1:t}) := \sum_{\rho \in \mathcal{M}_U} 2^{-K(\rho)} \rho(x_{1:t} \,|\, a_{1:t}), \tag{12}
$$

Theorem 1 shows for all $n \in \mathbb{N}$ and for all $a_{1:n}$,

$$
\sum_{k=1}^{n} \sum_{x_{1:k}} \mu(x_{<k} \,|\, a_{<k}) \Big(\mu(x_k \,|\, ax_{<k}a_k) - \xi_U(x_k \,|\, ax_{<k}a_k) \Big)^2 \leq K(\mu) \ln 2. \tag{13}
$$

3.4 Direct AIXI Approximation

We are now in a position to describe our approach to AIXI approximation. For prediction, we seek a computationally efficient mixture environment model ξ as a replacement for ξ_U. Ideally, ξ will retain ξ_U's bias towards simplicity and some of its generality. This will be achieved by placing a suitable Ockham prior over a set of candidate environment models.

For planning, we seek a scalable algorithm that can, given a limited set of resources, compute an approximation to the expectimax action given by

$$a^*_{t+1} = \arg\max_{a_{t+1}} V^m_{\xi_U}(ax_{1:t}a_{t+1}).$$

The main difficulties are of course computational. The next two sections introduce two algorithms that can be used to (partially) fulfill these criteria. Their subsequent combination will constitute our AIXI approximation.

4. Expectimax Approximation with Monte-Carlo Tree Search

Naïve computation of the expectimax operation (Equation 6) takes $O(|\mathcal{A} \times \mathcal{X}|^m)$ time, unacceptable for all but tiny values of m. This section introduces ρUCT, a generalisation of the popular Monte-Carlo Tree Search algorithm UCT (Kocsis & Szepesvári, 2006), that can be used to approximate a finite horizon expectimax operation given an environment model ρ. As an environment model subsumes both MDPs and POMDPs, ρUCT effectively extends the UCT algorithm to a wider class of problem domains.

4.1 Background

UCT has proven particularly effective in dealing with difficult problems containing large state spaces. It requires a generative model that when given a state-action pair (s, a) produces a subsequent state-reward pair (s', r) distributed according to $\Pr(s', r \mid s, a)$. By successively sampling trajectories through the state space, the UCT algorithm incrementally constructs a search tree, with each node containing an estimate of the value of each state. Given enough time, these estimates converge to their true values.

The ρUCT algorithm can be realised by replacing the notion of state in UCT by an agent history h (which is always a sufficient statistic) and using an environment model ρ to predict the next percept. The main subtlety with this extension is that now the history condition of the percept probability $\rho(or \mid h)$ needs to be updated during the search. This is to reflect the extra information an agent will have at a hypothetical future point in time. Furthermore, Proposition 1 allows ρUCT to be instantiated with a mixture environment model, which directly incorporates the model uncertainty of the agent into the planning process. This gives (in principle, provided that the model class contains the true environment and ignoring issues of limited computation) the well known Bayesian solution to the exploration/exploitation dilemma; namely, if a reduction in model uncertainty would lead to higher expected future reward, ρUCT would recommend an information gathering action.

4.2 Overview

ρUCT is a best-first Monte-Carlo Tree Search technique that iteratively constructs a search tree in memory. The tree is composed of two interleaved types of nodes: decision nodes and chance nodes. These correspond to the alternating max and sum operations in the expectimax operation. Each node in the tree corresponds to a history h. If h ends with an action, it is a chance node; if h ends with an observation-reward pair, it is a decision node. Each node contains a statistical estimate of the future reward.

Initially, the tree starts with a single decision node containing $|\mathcal{A}|$ children. Much like existing MCTS methods (Chaslot, Winands, Uiterwijk, van den Herik, & Bouzy, 2008a), there are four

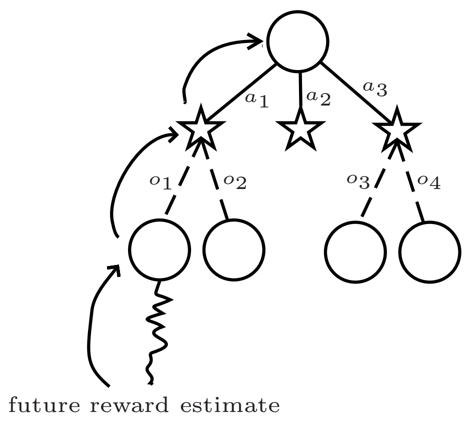

Figure 1: A ρUCT search tree

conceptual phases to a single iteration of ρUCT. The first is the *selection* phase, where the search tree is traversed from the root node to an existing leaf chance node *n*. The second is the *expansion* phase, where a new decision node is added as a child to *n*. The third is the *simulation* phase, where a rollout policy in conjunction with the environment model ρ is used to sample a possible future path from *n* until a fixed distance from the root is reached. Finally, the *backpropagation* phase updates the value estimates for each node on the reverse trajectory leading back to the root. Whilst time remains, these four conceptual operations are repeated. Once the time limit is reached, an approximate best action can be selected by looking at the value estimates of the children of the root node.

During the selection phase, action selection at decision nodes is done using a policy that balances exploration and exploitation. This policy has two main effects:

- to gradually move the estimates of the future reward towards the maximum attainable future reward if the agent acted optimally.
- to cause asymmetric growth of the search tree towards areas that have high predicted reward, implicitly pruning large parts of the search space.

The future reward at leaf nodes is estimated by choosing actions according to a heuristic policy until a total of *m* actions have been made by the agent, where *m* is the search horizon. This heuristic estimate helps the agent to focus its exploration on useful parts of the search tree, and in practice allows for a much larger horizon than a brute-force expectimax search.

ρUCT builds a sparse search tree in the sense that observations are only added to chance nodes once they have been generated along some sample path. A full-width expectimax search tree would not be sparse; each possible stochastic outcome would be represented by a distinct node in the search tree. For expectimax, the branching factor at chance nodes is thus $|\mathcal{O}|$, which means that searching to even moderate sized *m* is intractable.

Figure 1 shows an example ρUCT tree. Chance nodes are denoted with stars. Decision nodes are denoted by circles. The dashed lines from a star node indicate that not all of the children have been expanded. The squiggly line at the base of the leftmost leaf denotes the execution of a rollout policy. The arrows proceeding up from this node indicate the flow of information back up the tree; this is defined in more detail below.

4.3 Action Selection at Decision Nodes

A decision node will always contain $|\mathcal{A}|$ distinct children, all of whom are chance nodes. Associated with each decision node representing a particular history h will be a value function estimate, $\hat{V}(h)$. During the selection phase, a child will need to be picked for further exploration. Action selection in MCTS poses a classic exploration/exploitation dilemma. On one hand we need to allocate enough visits to all children to ensure that we have accurate estimates for them, but on the other hand we need to allocate enough visits to the maximal action to ensure convergence of the node to the value of the maximal child node.

Like UCT, ρUCT recursively uses the UCB policy (Auer, 2002) from the n-armed bandit setting at each decision node to determine which action needs further exploration. Although the uniform logarithmic regret bound no longer carries across from the bandit setting, the UCB policy has been shown to work well in practice in complex domains such as computer Go (Gelly & Wang, 2006) and General Game Playing (Finnsson & Björnsson, 2008). This policy has the advantage of ensuring that at each decision node, every action eventually gets explored an infinite number of times, with the best action being selected exponentially more often than actions of lesser utility.

Definition 5. *The visit count $T(h)$ of a decision node h is the number of times h has been sampled by the ρUCT algorithm. The visit count of the chance node found by taking action a at h is defined similarly, and is denoted by $T(ha)$.*

Definition 6. *Suppose m is the remaining search horizon and each instantaneous reward is bounded in the interval $[\alpha, \beta]$. Given a node representing a history h in the search tree, the action picked by the UCB action selection policy is:*

$$a_{UCB}(h) := \arg\max_{a \in \mathcal{A}} \begin{cases} \frac{1}{m(\beta-\alpha)} \hat{V}(ha) + C \sqrt{\frac{\log(T(h))}{T(ha)}} & \text{if } T(ha) > 0; \\ \infty & \text{otherwise}, \end{cases} \tag{14}$$

where $C \in \mathbb{R}$ is a positive parameter that controls the ratio of exploration to exploitation. If there are multiple maximal actions, one is chosen uniformly at random.

Note that we need a linear scaling of $\hat{V}(ha)$ in Definition 6 because the UCB policy is only applicable for rewards confined to the $[0, 1]$ interval.

4.4 Chance Nodes

Chance nodes follow immediately after an action is selected from a decision node. Each chance node ha following a decision node h contains an estimate of the future utility denoted by $\hat{V}(ha)$. Also associated with the chance node ha is a density $\rho(\cdot \,|\, ha)$ over observation-reward pairs.

After an action a is performed at node h, $\rho(\cdot \,|\, ha)$ is sampled once to generate the next observation-reward pair or. If or has not been seen before, the node $haor$ is added as a child of ha.

4.5 Estimating Future Reward at Leaf Nodes

If a leaf decision node is encountered at depth $k < m$ in the tree, a means of estimating the future reward for the remaining $m - k$ time steps is required. MCTS methods use a heuristic rollout policy Π to estimate the sum of future rewards $\sum_{i=k}^{m} r_i$. This involves sampling an action a from $\Pi(h)$, sampling a percept or from $\rho(\cdot \mid ha)$, appending aor to the current history h and then repeating this process until the horizon is reached. This procedure is described in Algorithm 4. A natural baseline policy is Π_{random}, which chooses an action uniformly at random at each time step.

As the number of simulations tends to infinity, the structure of the ρUCT search tree converges to the full depth m expectimax tree. Once this occurs, the rollout policy is no longer used by ρUCT. This implies that the asymptotic value function estimates of ρUCT are invariant to the choice of Π. In practice, when time is limited, not enough simulations will be performed to grow the full expectimax tree. Therefore, the choice of rollout policy plays an important role in determining the overall performance of ρUCT. Methods for learning Π online are discussed as future work in Section 9. Unless otherwise stated, all of our subsequent results will use Π_{random}.

4.6 Reward Backup

After the selection phase is completed, a path of nodes $n_1 n_2 \ldots n_k$, $k \leq m$, will have been traversed from the root of the search tree n_1 to some leaf n_k. For each $1 \leq j \leq k$, the statistics maintained for history h_{n_j} associated with node n_j will be updated as follows:

$$\hat{V}(h_{n_j}) \leftarrow \frac{T(h_{n_j})}{T(h_{n_j})+1}\hat{V}(h_{n_j}) + \frac{1}{T(h_{n_j})+1}\sum_{i=j}^{m} r_i \tag{15}$$

$$T(h_{n_j}) \leftarrow T(h_{n_j}) + 1 \tag{16}$$

Equation (15) computes the mean return. Equation (16) increments the visit counter. Note that the same backup operation is applied to both decision and chance nodes.

4.7 Pseudocode

The pseudocode of the ρUCT algorithm is now given.

After a percept has been received, Algorithm 1 is invoked to determine an approximate best action. A *simulation* corresponds to a single call to SAMPLE from Algorithm 1. By performing a number of simulations, a search tree Ψ whose root corresponds to the current history h is constructed. This tree will contain estimates $\hat{V}^m_\rho(ha)$ for each $a \in \mathcal{A}$. Once the available thinking time is exceeded, a maximising action $\hat{a}^*_h := \arg\max_{a\in\mathcal{A}} \hat{V}^m_\rho(ha)$ is retrieved by BESTACTION. Importantly, Algorithm 1 is *anytime*, meaning that an approximate best action is always available. This allows the agent to effectively utilise all available computational resources for each decision.

Algorithm 1 ρUCT(h, m)

```
Require: A history $h$
Require: A search horizon $m \in \mathbb{N}$
1: INITIALISE($\Psi$)
2: repeat
3:     SAMPLE($\Psi, h, m$)
4: until out of time
5: return BESTACTION($\Psi, h$)
```

For simplicity of exposition, INITIALISE can be understood to simply clear the entire search tree Ψ. In practice, it is possible to carry across information from one time step to another. If Ψ_t is the search tree obtained at the end of time t, and aor is the agent's actual action and experience at time t, then we can keep the subtree rooted at node $\Psi_t(hao)$ in Ψ_t and make that the search tree Ψ_{t+1} for use at the beginning of the next time step. The remainder of the nodes in Ψ_t can then be deleted.

Algorithm 2 describes the recursive routine used to sample a single future trajectory. It uses the SELECTACTION routine to choose moves at decision nodes, and invokes the ROLLOUT routine at unexplored leaf nodes. The ROLLOUT routine picks actions according to the rollout policy Π until the (remaining) horizon is reached, returning the accumulated reward. After a complete trajectory of length m is simulated, the value estimates are updated for each node traversed as per Section 4.6. Notice that the recursive calls on Lines 6 and 11 append the most recent percept or action to the history argument.

Algorithm 2 SAMPLE(Ψ, h, m)

```
Require: A search tree $\Psi$
Require: A history $h$
Require: A remaining search horizon $m \in \mathbb{N}$
 1: if $m = 0$ then
 2:     return 0
 3: else if $\Psi(h)$ is a chance node then
 4:     Generate $(o, r)$ from $\rho(or \mid h)$
 5:     Create node $\Psi(hor)$ if $T(hor) = 0$
 6:     reward ← $r$ + SAMPLE($\Psi, hor, m - 1$)
 7: else if $T(h) = 0$ then
 8:     reward ← ROLLOUT($h, m$)
 9: else
10:     $a$ ← SELECTACTION($\Psi, h$)
11:     reward ← SAMPLE($\Psi, ha, m$)
12: end if
13: $\hat{V}(h) \leftarrow \frac{1}{T(h)+1}[reward + T(h)\hat{V}(h)]$
14: $T(h) \leftarrow T(h) + 1$
15: return reward
```

The action chosen by SELECTACTION is specified by the UCB policy described in Definition 6. If the selected child has not been explored before, a new node is added to the search tree. The constant C is a parameter that is used to control the shape of the search tree; lower values of C create deep, selective search trees, whilst higher values lead to shorter, bushier trees. UCB automatically focuses attention on the best looking action in such a way that the sample estimate $\hat{V}_\rho(h)$ converges to $V_\rho(h)$, whilst still exploring alternate actions sufficiently often to guarantee that the best action will be eventually found.

4.8 Consistency of ρUCT

Let μ be the true underlying environment. We now establish the link between the expectimax value $V_\mu^m(h)$ and its estimate $\hat{V}_\mu^m(h)$ computed by the ρUCT algorithm.

Kocsis and Szepesvári (2006) show that with an appropriate choice of C, the UCT algorithm is consistent in finite horizon MDPs. By interpreting histories as Markov states, our general agent

Algorithm 3 SelectAction(Ψ, h)

```
Require: A search tree $\Psi$
Require: A history $h$
Require: An exploration/exploitation constant $C$
1: $\mathcal{U} = \{a \in \mathcal{A} \colon T(ha) = 0\}$
2: if $\mathcal{U} \neq \{\}$ then
3:     Pick $a \in \mathcal{U}$ uniformly at random
4:     Create node $\Psi(ha)$
5:     return a
6: else
7:     return $\arg\max_{a \in \mathcal{A}} \left\{ \frac{1}{m(\beta-\alpha)} \hat{V}(ha) + C \sqrt{\frac{\log(T(h))}{T(ha)}} \right\}$
8: end if
```

Algorithm 4 Rollout(h, m)

```
Require: A history $h$
Require: A remaining search horizon $m \in \mathbb{N}$
Require: A rollout function $\Pi$
1: $reward \leftarrow 0$
2: for $i = 1$ to $m$ do
3:     Generate $a$ from $\Pi(h)$
4:     Generate $(o, r)$ from $\rho(or \,|\, ha)$
5:     $reward \leftarrow reward + r$
6:     $h \leftarrow haor$
7: end for
8: return reward
```

problem reduces to a finite horizon MDP. This means that the results of Kocsis and Szepesvári (2006) are now directly applicable. Restating the main consistency result in our notation, we have

$$\forall \epsilon \forall h \lim_{T(h)\to\infty} \Pr\left(|V^m_\mu(h) - \hat{V}^m_\mu(h)| \leq \epsilon \right) = 1, \tag{17}$$

that is, $\hat{V}^m_\mu(h) \to V^m_\mu(h)$ with probability 1. Furthermore, the probability that a suboptimal action (with respect to $V^m_\mu(\cdot)$) is picked by ρUCT goes to zero in the limit. Details of this analysis can be found in the work of Kocsis and Szepesvári (2006).

4.9 Parallel Implementation of ρUCT

As a Monte-Carlo Tree Search routine, Algorithm 1 can be easily parallelised. The main idea is to concurrently invoke the Sample routine whilst providing appropriate locking mechanisms for the interior nodes of the search tree. A highly scalable parallel implementation is beyond the scope of the paper, but it is worth noting that ideas applicable to high performance Monte-Carlo Go programs (Chaslot, Winands, & Herik, 2008b) can be easily transferred to our setting.

5. Model Class Approximation using Context Tree Weighting

We now turn our attention to the construction of an efficient mixture environment model suitable for the general reinforcement learning problem. If computation were not an issue, it would be sufficient to first specify a large model class $\mathcal{M}$, and then use Equations (8) or (3.1) for online prediction. The problem with this approach is that at least $O(|\mathcal{M}|)$ time is required to process each new piece of experience. This is simply too slow for the enormous model classes required by general agents. Instead, this section will describe how to predict in $O(\log\log|\mathcal{M}|)$ time, using a mixture environment model constructed from an adaptation of the Context Tree Weighting algorithm.

5.1 Context Tree Weighting

Context Tree Weighting (CTW) (Willems et al., 1995; Willems, Shtarkov, & Tjalkens, 1997) is an efficient and theoretically well-studied binary sequence prediction algorithm that works well in practice (Begleiter, El-Yaniv, & Yona, 2004). It is an online Bayesian model averaging algorithm that computes, at each time point t, the probability

$$\Pr(y_{1:t}) = \sum_{M} \Pr(M) \Pr(y_{1:t} \,|\, M), \tag{18}$$

where $y_{1:t}$ is the binary sequence seen so far, M is a prediction suffix tree (Rissanen, 1983; Ron, Singer, & Tishby, 1996), $\Pr(M)$ is the prior probability of M, and the summation is over *all* prediction suffix trees of bounded depth D. This is a huge class, covering all D-order Markov processes. A naïve computation of (18) takes time $O(2^{2^D})$; using CTW, this computation requires only $O(D)$ time. In this section, we outline two ways in which CTW can be generalised to compute probabilities of the form

$$\Pr(x_{1:t} \,|\, a_{1:t}) = \sum_{M} \Pr(M) \Pr(x_{1:t} \,|\, M, a_{1:t}), \tag{19}$$

where $x_{1:t}$ is a percept sequence, $a_{1:t}$ is an action sequence, and M is a prediction suffix tree as in (18). These generalisations will allow CTW to be used as a mixture environment model.

5.2 Krichevsky-Trofimov Estimator

We start with a brief review of the KT estimator (Krichevsky & Trofimov, 1981) for Bernoulli distributions. Given a binary string $y_{1:t}$ with a zeros and b ones, the KT estimate of the probability of the next symbol is as follows:

$$\Pr_{kt}(Y_{t+1} = 1 \,|\, y_{1:t}) := \frac{b + 1/2}{a + b + 1} \tag{20}$$

$$\Pr_{kt}(Y_{t+1} = 0 \,|\, y_{1:t}) := 1 - \Pr_{kt}(Y_{t+1} = 1 \,|\, y_{1:t}). \tag{21}$$

The KT estimator is obtained via a Bayesian analysis by putting an uninformative (Jeffreys Beta(1/2,1/2)) prior $\Pr(\theta) \propto \theta^{-1/2}(1-\theta)^{-1/2}$ on the parameter $\theta \in [0,1]$ of the Bernoulli distribution. From (20)-(21), we obtain the following expression for the block probability of a string:

$$\begin{aligned} \Pr_{kt}(y_{1:t}) &= \Pr_{kt}(y_1 \,|\, \epsilon)\Pr_{kt}(y_2 \,|\, y_1) \cdots \Pr_{kt}(y_t \,|\, y_{<t}) \\ &= \textstyle\int \theta^b (1-\theta)^a \Pr(\theta)\, d\theta. \end{aligned}$$

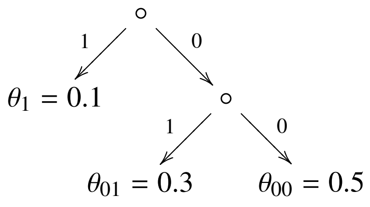

Figure 2: An example prediction suffix tree

Since $\Pr_{kt}(s)$ depends only on the number of zeros a_s and ones b_s in a string s, if we let 0^a1^b denote a string with a zeroes and b ones, then we have

$$\Pr_{kt}(s) = \Pr_{kt}(0^{a_s}1^{b_s}) = \frac{1/2(1+1/2)\cdots(a_s-1/2)1/2(1+1/2)\cdots(b_s-1/2)}{(a_s+b_s)!}. \tag{22}$$

We write $\Pr_{kt}(a,b)$ to denote $\Pr_{kt}(0^a1^b)$ in the following. The quantity $\Pr_{kt}(a,b)$ can be updated incrementally (Willems et al., 1995) as follows:

$$\Pr_{kt}(a+1,b) = \frac{a+1/2}{a+b+1}\Pr_{kt}(a,b) \tag{23}$$
$$\Pr_{kt}(a,b+1) = \frac{b+1/2}{a+b+1}\Pr_{kt}(a,b), \tag{24}$$

with the base case being $\Pr_{kt}(0,0) = 1$.

5.3 Prediction Suffix Trees

We next describe prediction suffix trees, which are a form of variable-order Markov models.

In the following, we work with binary trees where all the left edges are labeled 1 and all the right edges are labeled 0. Each node in such a binary tree M can be identified by a string in $\{0,1\}^*$ as follows: ϵ represents the root node of M; and if $n \in \{0,1\}^*$ is a node in M, then $n1$ and $n0$ represent the left and right child of node n respectively. The set of M's leaf nodes $L(M) \subset \{0,1\}^*$ form a complete prefix-free set of strings. Given a binary string $y_{1:t}$ such that $t \geq$ the depth of M, we define $M(y_{1:t}) := y_t y_{t-1} \ldots y_{t'}$, where $t' \leq t$ is the (unique) positive integer such that $y_t y_{t-1} \ldots y_{t'} \in L(M)$. In other words, $M(y_{1:t})$ represents the suffix of $y_{1:t}$ that occurs in tree M.

Definition 7. *A prediction suffix tree (PST) is a pair (M,Θ), where M is a binary tree and associated with each leaf node l in M is a probability distribution over $\{0,1\}$ parametrised by $\theta_l \in \Theta$. We call M the model of the PST and Θ the parameter of the PST, in accordance with the terminology of Willems et al. (1995).*

A prediction suffix tree (M,Θ) maps each binary string $y_{1:t}$, where $t \geq$ the depth of M, to the probability distribution $\theta_{M(y_{1:t})}$; the intended meaning is that $\theta_{M(y_{1:t})}$ is the probability that the next bit following $y_{1:t}$ is 1. For example, the PST shown in Figure 2 maps the string 1110 to $\theta_{M(1110)} = \theta_{01} = 0.3$, which means the next bit after 1110 is 1 with probability 0.3.

In practice, to use prediction suffix trees for binary sequence prediction, we need to learn both the model and parameter of a prediction suffix tree from data. We will deal with the model-learning part later. Assuming the model of a PST is known/given, the parameter of the PST can be learnt using the KT estimator as follows. We start with $\theta_l := \Pr_{kt}(1 \,|\, \epsilon) = 1/2$ at each leaf node l of M. If

d is the depth of M, then the first d bits $y_{1:d}$ of the input sequence are set aside for use as an initial context and the variable h denoting the bit sequence seen so far is set to $y_{1:d}$. We then repeat the following steps as long as needed:

1. predict the next bit using the distribution $\theta_{M(h)}$;
2. observe the next bit y, update $\theta_{M(h)}$ using Formula (20) by incrementing either a or b according to the value of y, and then set $h := hy$.

5.4 Action-Conditional PST

The above describes how a PST is used for binary sequence prediction. In the agent setting, we reduce the problem of predicting history sequences with general non-binary alphabets to that of predicting the bit representations of those sequences. Furthermore, we only ever condition on actions. This is achieved by appending bit representations of actions to the input sequence without a corresponding update of the KT estimators. These ideas are now formalised.

For convenience, we will assume without loss of generality that $|\mathcal{A}| = 2^{l_{\mathcal{A}}}$ and $|\mathcal{X}| = 2^{l_{\mathcal{X}}}$ for some $l_{\mathcal{A}}, l_{\mathcal{X}} > 0$. Given $a \in \mathcal{A}$, we denote by $[\![a]\!] = a[1, l_{\mathcal{A}}] = a[1]a[2] \dots a[l_{\mathcal{A}}] \in \{0,1\}^{l_{\mathcal{A}}}$ the bit representation of a. Observation and reward symbols are treated similarly. Further, the bit representation of a symbol sequence $x_{1:t}$ is denoted by $[\![x_{1:t}]\!] = [\![x_1]\!][\![x_2]\!] \dots [\![x_t]\!]$.

To do action-conditional sequence prediction using a PST with a given model M, we again start with $\theta_l := \Pr_{kt}(1 \,|\, \epsilon) = 1/2$ at each leaf node l of M. We also set aside a sufficiently long initial portion of the binary history sequence corresponding to the first few cycles to initialise the variable h as usual. The following steps are then repeated as long as needed:

1. set $h := h[\![a]\!]$, where a is the current selected action;
2. for $i := 1$ to $l_{\mathcal{X}}$ do
   (a) predict the next bit using the distribution $\theta_{M(h)}$;
   (b) observe the next bit $x[i]$, update $\theta_{M(h)}$ using Formula (20) according to the value of $x[i]$, and then set $h := hx[i]$.

Let M be the model of a prediction suffix tree, $a_{1:t} \in \mathcal{A}^t$ an action sequence, $x_{1:t} \in \mathcal{X}^t$ an observation-reward sequence, and $h := [\![ax_{1:t}]\!]$. For each node n in M, define $h_{M,n}$ by

$$h_{M,n} := h_{i_1} h_{i_2} \cdots h_{i_k} \tag{25}$$

where $1 \le i_1 < i_2 < \cdots < i_k \le t$ and, for each i, $i \in \{i_1, i_2, \dots i_k\}$ iff h_i is an observation-reward bit and n is a prefix of $M(h_{1:i-1})$. In other words, $h_{M,n}$ consists of all the observation-reward bits with context n. Thus we have the following expression for the probability of $x_{1:t}$ given M and $a_{1:t}$:

$$\begin{aligned}\Pr(x_{1:t} \,|\, M, a_{1:t}) &= \prod_{i=1}^{t} \Pr(x_i \,|\, M, ax_{<i}a_i) \\ &= \prod_{i=1}^{t} \prod_{j=1}^{l_{\mathcal{X}}} \Pr(x_i[j] \,|\, M, [\![ax_{<i}a_i]\!] x_i[1, j-1]) \\ &= \prod_{n \in L(M)} \Pr_{kt}(h_{M,n}). \end{aligned} \tag{26}$$

The last step follows by grouping the individual probability terms according to the node $n \in L(M)$ in which each bit falls and then observing Equation (22). The above deals with action-conditional prediction using a single PST. We now show how we can perform efficient action-conditional prediction using a Bayesian mixture of PSTs. First we specify a prior over PST models.

5.5 A Prior on Models of PSTs

Our prior $\Pr(M) := 2^{-\Gamma_D(M)}$ is derived from a natural prefix coding of the tree structure of a PST. The coding scheme works as follows: given a model of a PST of maximum depth D, a pre-order traversal of the tree is performed. Each time an internal node is encountered, we write down 1. Each time a leaf node is encountered, we write a 0 if the depth of the leaf node is less than D; otherwise we write nothing. For example, if $D = 3$, the code for the model shown in Figure 2 is 10100; if $D = 2$, the code for the same model is 101. The cost $\Gamma_D(M)$ of a model M is the length of its code, which is given by the number of nodes in M minus the number of leaf nodes in M of depth D. One can show that

$$\sum_{M \in C_D} 2^{-\Gamma_D(M)} = 1,$$

where C_D is the set of all models of prediction suffix trees with depth at most D; i.e. the prefix code is complete. We remark that the above is another way of describing the coding scheme in Willems et al. (1995). Note that this choice of prior imposes an Ockham-like penalty on large PST structures.

5.6 Context Trees

The following data structure is a key ingredient of the Action-Conditional CTW algorithm.

Definition 8. *A context tree of depth D is a perfect binary tree of depth D such that attached to each node (both internal and leaf) is a probability on* $\{0, 1\}^*$.

The node probabilities in a context tree are estimated from data by using a KT estimator at each node. The process to update a context tree with a history sequence is similar to a PST, except that:

1. the probabilities at each node in the path from the root to a leaf traversed by an observed bit are updated; and
2. we maintain block probabilities using Equations (22) to (24) instead of conditional probabilities.

This process can be best understood with an example. Figure 3 (left) shows a context tree of depth two. For expositional reasons, we show binary sequences at the nodes; the node probabilities are computed from these. Initially, the binary sequence at each node is empty. Suppose 1001 is the history sequence. Setting aside the first two bits 10 as an initial context, the tree in the middle of Figure 3 shows what we have after processing the third bit 0. The tree on the right is the tree we have after processing the fourth bit 1. In practice, we of course only have to store the counts of zeros and ones instead of complete subsequences at each node because, as we saw earlier in (22), $\Pr_{kt}(s) = \Pr_{kt}(a_s, b_s)$. Since the node probabilities are completely determined by the input sequence, we shall henceforth speak unambiguously about *the* context tree after seeing a sequence.

The context tree of depth D after seeing a sequence h has the following important properties:

1. the model of every PST of depth at most D can be obtained from the context tree by pruning off appropriate subtrees and treating them as leaf nodes;

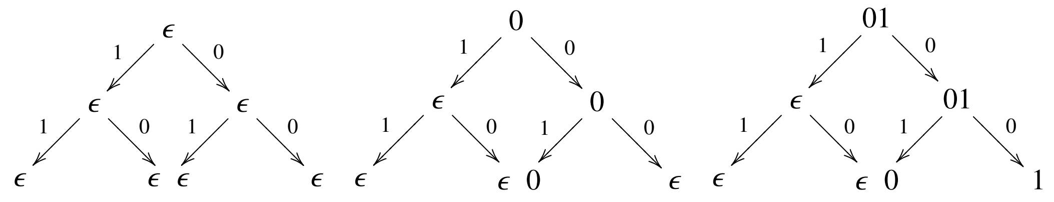

Figure 3: A depth-2 context tree (left); trees after processing two bits (middle and right)

2. the block probability of h as computed by each PST of depth at most D can be obtained from the node probabilities of the context tree via Equation (26).

These properties, together with an application of the distributive law, form the basis of the highly efficient Action Conditional CTW algorithm. We now formalise these insights.

5.7 Weighted Probabilities

The weighted probability P_w^n of each node n in the context tree T after seeing $h := [\![ax_{1:t}]\!]$ is defined inductively as follows:

$$P_w^n := \begin{cases} \Pr_{kt}(h_{T,n}) & \text{if } n \text{ is a leaf node;} \\ \frac{1}{2}\Pr_{kt}(h_{T,n}) + \frac{1}{2}P_w^{n0} \times P_w^{n1} & \text{otherwise,} \end{cases} \tag{27}$$

where $h_{T,n}$ is as defined in (25).

Lemma 2 (Willems et al., 1995)**.** *Let T be the depth-D context tree after seeing $h := [\![ax_{1:t}]\!]$. For each node n in T at depth d, we have*

$$P_w^n = \sum_{M \in C_{D-d}} 2^{-\Gamma_{D-d}(M)} \prod_{n' \in L(M)} \Pr_{kt}(h_{T,nn'}). \tag{28}$$

Proof. The proof proceeds by induction on d. The statement is clearly true for the leaf nodes at depth D. Assume now the statement is true for all nodes at depth $d+1$, where $0 \le d < D$. Consider a node n at depth d. Letting $\overline{d} = D - d$, we have

$$\begin{aligned} P_w^n &= \frac{1}{2}\Pr_{kt}(h_{T,n}) + \frac{1}{2}P_w^{n0}P_w^{n1} \\ &= \frac{1}{2}\Pr_{kt}(h_{T,n}) + \frac{1}{2}\left[\sum_{M \in C_{\overline{d+1}}} 2^{-\Gamma_{\overline{d+1}}(M)} \prod_{n' \in L(M)} \Pr_{kt}(h_{T,n0n'})\right]\left[\sum_{M \in C_{\overline{d+1}}} 2^{-\Gamma_{\overline{d+1}}(M)} \prod_{n' \in L(M)} \Pr_{kt}(h_{T,n1n'})\right] \\ &= \frac{1}{2}\Pr_{kt}(h_{T,n}) + \sum_{M_1 \in C_{\overline{d+1}}} \sum_{M_2 \in C_{\overline{d+1}}} 2^{-(\Gamma_{\overline{d+1}}(M_1) + \Gamma_{\overline{d+1}}(M_2) + 1)} \left[\prod_{n' \in L(M_1)} \Pr_{kt}(h_{T,n0n'})\right]\left[\prod_{n' \in L(M_2)} \Pr_{kt}(h_{T,n1n'})\right] \\ &= \frac{1}{2}\Pr_{kt}(h_{T,n}) + \sum_{\widehat{M_1M_2} \in C_{\overline{d}}} 2^{-\Gamma_{\overline{d}}(\widehat{M_1M_2})} \prod_{n' \in L(\widehat{M_1M_2})} \Pr_{kt}(h_{T,nn'}) \\ &= \sum_{M \in C_{D-d}} 2^{-\Gamma_{D-d}(M)} \prod_{n' \in L(M)} \Pr_{kt}(h_{T,nn'}), \end{aligned}$$

where $\widehat{M_1M_2}$ denotes the tree in $C_{\overline{d}}$ whose left and right subtrees are M_1 and M_2 respectively. □

5.8 Action Conditional CTW as a Mixture Environment Model

A corollary of Lemma 2 is that at the root node ϵ of the context tree T after seeing $h := [\![ax_{1:t}]\!]$, we have

$$P_w^\epsilon = \sum_{M \in C_D} 2^{-\Gamma_D(M)} \prod_{l \in L(M)} \Pr_{kt}(h_{T,l}) \tag{29}$$

$$= \sum_{M \in C_D} 2^{-\Gamma_D(M)} \prod_{l \in L(M)} \Pr_{kt}(h_{M,l}) \tag{30}$$

$$= \sum_{M \in C_D} 2^{-\Gamma_D(M)} \Pr(x_{1:t} \,|\, M, a_{1:t}), \tag{31}$$

where the last step follows from Equation (26). Equation (31) shows that the quantity computed by the Action-Conditional CTW algorithm is exactly a mixture environment model. Note that the conditional probability is always defined, as CTW assigns a non-zero probability to any sequence. To sample from this conditional probability, we simply sample the individual bits of x_t one by one.

In summary, to do prediction using Action-Conditional CTW, we set aside a sufficiently long initial portion of the binary history sequence corresponding to the first few cycles to initialise the variable h and then repeat the following steps as long as needed:

1. set $h := h[\![a]\!]$, where a is the current selected action;
2. for $i := 1$ to $l_\mathcal{X}$ do
   (a) predict the next bit using the weighted probability P_w^ϵ;
   (b) observe the next bit $x[i]$, update the context tree using h and $x[i]$, calculate the new weighted probability P_w^ϵ, and then set $h := hx[i]$.

5.9 Incorporating Type Information

One drawback of the Action-Conditional CTW algorithm is the potential loss of type information when mapping a history string to its binary encoding. This type information may be needed for predicting well in some domains. Although it is always possible to choose a binary encoding scheme so that the type information can be inferred by a depth limited context tree, it would be desirable to remove this restriction so that our agent can work with arbitrary encodings of the percept space.

One option would be to define an action-conditional version of multi-alphabet CTW (Tjalkens, Shtarkov, & Willems, 1993), with the alphabet consisting of the entire percept space. The downside of this approach is that we then lose the ability to exploit the structure within each percept. This can be critical when dealing with large observation spaces, as noted by McCallum (1996). The key difference between his U-Tree and USM algorithms is that the former could discriminate between individual components within an observation, whereas the latter worked only at the symbol level. As we shall see in Section 7, this property can be helpful when dealing with larger problems.

Fortunately, it is possible to get the best of both worlds. We now describe a technique that incorporates type information whilst still working at the bit level. The trick is to chain together $k := l_\mathcal{X}$ action conditional PSTs, one for each bit of the percept space, with appropriately overlapping binary contexts. More precisely, given a history h, the context for the ith PST is the most recent $D+i-1$ bits of the bit-level history string $[\![h]\!]x[1, i-1]$. To ensure that each percept bit is dependent on the same portion of h, $D+i-1$ (instead of only D) bits are used. Thus if we denote the PST model for the ith bit in a percept x by M_i, and the joint model by M, we now have:

$$\begin{aligned}\Pr(x_{1:t} \,|\, M, a_{1:t}) &= \prod_{i=1}^{t} \Pr(x_i \,|\, M, ax_{<i}a_i) \\ &= \prod_{i=1}^{t} \prod_{j=1}^{k} \Pr(x_i[j] \,|\, M_j, [\![ax_{<i}a_i]\!]x_i[1, j-1]) \qquad (32) \\ &= \prod_{j=1}^{k} \Pr(x_{1:t}[j] \,|\, M_j, x_{1:t}[-j], a_{1:t})\end{aligned}$$

where $x_{1:t}[i]$ denotes $x_1[i]x_2[i] \dots x_t[i]$, $x_{1:t}[-i]$ denotes $x_1[-i]x_2[-i] \dots x_t[-i]$, with $x_t[-j]$ denoting $x_t[1] \dots x_t[j-1]x_t[j+1] \dots x_t[k]$. The last step follows by swapping the two products in (32) and using the above notation to refer to the product of probabilities of the jth bit in each percept x_i, for $1 \leq i \leq t$.

We next place a prior on the space of factored PST models $M \in C_D \times \dots \times C_{D+k-1}$ by assuming that each factor is independent, giving

$$\Pr(M) = \Pr(M_1, \dots, M_k) = \prod_{i=1}^{k} 2^{-\Gamma_{D_i}(M_i)} = 2^{-\sum_{i=1}^{k} \Gamma_{D_i}(M_i)},$$

where $D_i := D + i - 1$. This induces the following mixture environment model

$$\xi(x_{1:t} \,|\, a_{1:t}) := \sum_{M \in C_{D_1} \times \dots \times C_{D_k}} 2^{-\sum_{i=1}^{k} \Gamma_{D_i}(M_i)} \Pr(x_{1:t} \,|\, M, a_{1:t}). \qquad (33)$$

This can now be rearranged into a product of efficiently computable mixtures, since

$$\begin{aligned}\xi(x_{1:t} \,|\, a_{1:t}) &= \sum_{M_1 \in C_{D_1}} \cdots \sum_{M_k \in C_{D_k}} 2^{-\sum_{i=1}^{k} \Gamma_{D_i}(M_i)} \prod_{j=1}^{k} \Pr(x_{1:t}[j] \,|\, M_j, x_{1:t}[-j], a_{1:t}) \\ &= \prod_{j=1}^{k} \left(\sum_{M_j \in C_{D_j}} 2^{-\Gamma_{D_j}(M_j)} \Pr(x_{1:t}[j] \,|\, M_j, x_{1:t}[-j], a_{1:t}) \right). \qquad (34)\end{aligned}$$

Note that for each factor within Equation (34), a result analogous to Lemma 2 can be established by appropriately modifying Lemma 2's proof to take into account that now only one bit per percept is being predicted. This leads to the following scheme for incrementally maintaining Equation (33):

1. Initialise $h \leftarrow \epsilon$, $t \leftarrow 1$. Create k context trees.
2. Determine action a_t. Set $h \leftarrow ha_t$.
3. Receive x_t. For each bit $x_t[i]$ of x_t, update the ith context tree with $x_t[i]$ using history $hx[1, i-1]$ and recompute P_w^ϵ using Equation (27).
4. Set $h \leftarrow hx_t$, $t \leftarrow t + 1$. Goto 2.

We will refer to this technique as Factored Action-Conditional CTW, or the FAC-CTW algorithm for short.

5.10 Convergence to the True Environment

We now show that FAC-CTW performs well in the class of stationary n-Markov environments. Importantly, this includes the class of Markov environments used in state-based reinforcement learning, where the most recent action/observation pair (a_t, x_{t-1}) is a sufficient statistic for the prediction of x_t.

Definition 9. *Given $n \in \mathbb{N}$, an environment μ is said to be n-Markov if for all $t > n$, for all $a_{1:t} \in \mathcal{A}^t$, for all $x_{1:t} \in \mathcal{X}^t$ and for all $h \in (\mathcal{A} \times \mathcal{X})^{t-n-1} \times \mathcal{A}$*

$$\mu(x_t \,|\, ax_{<t}a_t) = \mu(x_t \,|\, hx_{t-n}ax_{t-n+1:t-1}a_t). \tag{35}$$

Furthermore, an n-Markov environment is said to be stationary if for all $ax_{1:n}a_{n+1} \in (\mathcal{A} \times \mathcal{X})^n \times \mathcal{A}$, for all $h, h' \in (\mathcal{A} \times \mathcal{X})^*$,

$$\mu(\cdot \,|\, hax_{1:n}a_{n+1}) = \mu(\cdot \,|\, h'ax_{1:n}a_{n+1}). \tag{36}$$

It is easy to see that any stationary n-Markov environment can be represented as a product of sufficiently large, fixed parameter PSTs. Theorem 1 states that the predictions made by a mixture environment model only converge to those of the true environment when the model class contains a model sufficiently close to the true environment. However, no *stationary* n-Markov environment model is contained within the model class of FAC-CTW, since each model updates the parameters for its KT-estimators as more data is seen. Fortunately, this is not a problem, since this updating produces models that are sufficiently close to any stationary n-Markov environment for Theorem 1 to be meaningful.

Lemma 3. *If $\mathcal{M}$ is the model class used by* FAC-CTW *with a context depth D, μ is an environment expressible as a product of $k := l_{\mathcal{X}}$ fixed parameter PSTs $(M_1, \Theta_1), \ldots, (M_k, \Theta_k)$ of maximum depth D and $\rho(\cdot \,|\, a_{1:n}) \equiv \Pr(\cdot \,|\, (M_1, \ldots, M_k), a_{1:n}) \in \mathcal{M}$ then for all $n \in \mathbb{N}$, for all $a_{1:n} \in \mathcal{A}^n$,*

$$D_{1:n}(\mu \,\|\, \rho) \leq \sum_{j=1}^{k} |L(M_j)| \, \gamma\left(\frac{n}{|L(M_j)|}\right)$$

where

$$\gamma(z) := \begin{cases} z & \text{for} \quad 0 \leq z < 1 \\ \frac{1}{2}\log z + 1 & \text{for} \quad z \geq 1. \end{cases}$$

Proof. For all $n \in \mathbb{N}$, for all $a_{1:n} \in \mathcal{A}^n$,

$$\begin{aligned}
D_{1:n}(\mu \,\|\, \rho) &= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\mu(x_{1:n} \,|\, a_{1:n})}{\rho(x_{1:n} \,|\, a_{1:n})} \\
&= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \ln \frac{\prod_{j=1}^{k} \Pr(x_{1:n}[j] \,|\, M_j, \Theta_j, x_{1:n}[-j], a_{1:n})}{\prod_{j=1}^{k} \Pr(x_{1:n}[j] \,|\, M_j, x_{1:n}[-j], a_{1:n})} \\
&= \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \sum_{j=1}^{k} \ln \frac{\Pr(x_{1:n}[j] \,|\, M_j, \Theta_j, x_{1:n}[-j], a_{1:n})}{\Pr(x_{1:n}[j] \,|\, M_j, x_{1:n}[-j], a_{1:n})} \\
&\leq \sum_{x_{1:n}} \mu(x_{1:n} \,|\, a_{1:n}) \sum_{j=1}^{k} |L(M_j)| \gamma\left(\frac{n}{|L(M_j)|}\right)
\end{aligned} \tag{37}$$

$$= \sum_{j=1}^{k} |L(M_j)|\, \gamma\left(\frac{n}{|L(M_j)|}\right)$$

where $\Pr(x_{1:n}[j] \,|\, M_j, \Theta_j, x_{1:n}[-j], a_{1:n})$ denotes the probability of a fixed parameter PST (M_j, Θ_j) generating the sequence $x_{1:n}[j]$ and the bound introduced in (37) is from the work of Willems et al. (1995). □

If the unknown environment μ is stationary and n-Markov, Lemma 3 and Theorem 1 can be applied to the FAC-CTW mixture environment model ξ. Together they imply that the cumulative μ-expected squared difference between μ and ξ is bounded by $O(\log n)$. Also, the *per cycle* μ-expected squared difference between μ and ξ goes to zero at the rapid rate of $O(\log n/n)$. This allows us to conclude that FAC-CTW (with a sufficiently large context depth) will perform well on the class of stationary n-Markov environments.

5.11 Summary

We have described two different ways in which CTW can be extended to define a large and efficiently computable mixture environment model. The first is a complete derivation of the Action-Conditional CTW algorithm first presented in the work of Veness, Ng, Hutter, and Silver (2010). The second is the introduction of the FAC-CTW algorithm, which improves upon Action-Conditional CTW by automatically exploiting the type information available within the agent setting.

As the rest of the paper will make extensive use of the FAC-CTW algorithm, for clarity we define

$$\Upsilon(x_{1:t} \,|\, a_{1:t}) := \sum_{M \in C_{D_1} \times \cdots \times C_{D_k}} 2^{-\sum_{i=1}^{k} \Gamma_{D_i}(M_i)} \Pr(x_{1:t} \,|\, M, a_{1:t}). \tag{38}$$

Also recall that using Υ as a mixture environment model, the conditional probability of x_t given $ax_{<t}a_t$ is

$$\Upsilon(x_t \,|\, ax_{<t}a_t) = \frac{\Upsilon(x_{1:t} \,|\, a_{1:t})}{\Upsilon(x_{<t} \,|\, a_{<t})},$$

which follows directly from Equation (3). To generate a percept from this conditional probability distribution, we simply sample $l_{\mathcal{X}}$ bits, one by one, from Υ.

5.12 Relationship to AIXI

Before moving on, we examine the relationship between AIXI and our model class approximation. Using Υ in place of ρ in Equation (6), the optimal action for an agent at time t, having experienced $ax_{1:t-1}$, is given by

$$\begin{aligned}
a_t^* &= \arg\max_{a_t} \sum_{x_t} \frac{\Upsilon(x_{1:t} \,|\, a_{1:t})}{\Upsilon(x_{<t} \,|\, a_{<t})} \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \frac{\Upsilon(x_{1:t+m} \,|\, a_{1:t+m})}{\Upsilon(x_{<t+m} \,|\, a_{<t+m})} \left[\sum_{i=t}^{t+m} r_i\right] \\
&= \arg\max_{a_t} \sum_{x_t} \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \left[\sum_{i=t}^{t+m} r_i\right] \prod_{i=t}^{t+m} \frac{\Upsilon(x_{1:i} \,|\, a_{1:i})}{\Upsilon(x_{<i} \,|\, a_{<i})} \\
&= \arg\max_{a_t} \sum_{x_t} \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \left[\sum_{i=t}^{t+m} r_i\right] \frac{\Upsilon(x_{1:t+m} \,|\, a_{1:t+m})}{\Upsilon(x_{<t} \,|\, a_{<t})}
\end{aligned}$$

$$= \arg\max_{a_t} \sum_{x_t} \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \left[\sum_{i=t}^{t+m} r_i \right] \Upsilon(x_{1:t+m} \,|\, a_{1:t+m})$$

$$= \arg\max_{a_t} \sum_{x_t} \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \left[\sum_{i=t}^{t+m} r_i \right] \sum_{M \in C_{D_1} \times \cdots \times C_{D_k}} 2^{-\sum_{i=1}^{k} \Gamma_{D_i}(M_i)} \Pr(x_{1:t+m} \,|\, M, a_{1:t+m}). \tag{39}$$

Contrast (39) now with Equation (11) which we reproduce here:

$$a_t^* = \arg\max_{a_t} \sum_{x_t} \ldots \max_{a_{t+m}} \sum_{x_{t+m}} \left[\sum_{i=t}^{t+m} r_i \right] \sum_{\rho \in \mathcal{M}} 2^{-K(\rho)} \rho(x_{1:t+m} \,|\, a_{1:t+m}), \tag{40}$$

where $\mathcal{M}$ is the class of all enumerable chronological semimeasures, and $K(\rho)$ denotes the Kolmogorov complexity of ρ. The two expressions share a prior that enforces a bias towards simpler models. The main difference is in the subexpression describing the mixture over the model class. AIXI uses a mixture over all enumerable chronological semimeasures. This is scaled down to a (factored) mixture of prediction suffix trees in our setting. Although the model class used in AIXI is completely general, it is also incomputable. Our approximation has restricted the model class to gain the desirable computational properties of FAC-CTW.

6. Putting it All Together

Our approximate AIXI agent, MC-AIXI(fac-ctw), is realised by instantiating the ρUCT algorithm with $\rho = \Upsilon$. Some additional properties of this combination are now discussed.

6.1 Convergence of Value

We now show that using Υ in place of the true environment μ in the expectimax operation leads to good behaviour when μ is both stationary and n-Markov. This result combines Lemma 3 with an adaptation of the work of Hutter (2005, Thm. 5.36). For this analysis, we assume that the instantaneous rewards are non-negative (with no loss of generality), FAC-CTW is used with a sufficiently large context depth, the maximum life of the agent $b \in \mathbb{N}$ is fixed and that a bounded planning horizon $m_t := \min(H, b - t + 1)$ is used at each time t, with $H \in \mathbb{N}$ specifying the maximum planning horizon.

Theorem 2. *Using the* FAC-CTW *algorithm, for every policy π, if the true environment μ is expressible as a product of k PSTs $(M_1, \Theta_1), \ldots, (M_k, \Theta_k)$, for all $b \in \mathbb{N}$, we have*

$$\sum_{t=1}^{b} \mathbb{E}_{x_{<t} \sim \mu} \left[\left(v_\Upsilon^{m_t}(\pi, ax_{<t}) - v_\mu^{m_t}(\pi, ax_{<t}) \right)^2 \right] \leq 2H^3 r_{max}^2 \left[\sum_{i=1}^{k} \Gamma_{D_i}(M_i) + \sum_{j=1}^{k} |L(M_j)| \, \gamma\left(\frac{b}{|L(M_j)|} \right) \right]$$

where r_{max} is the maximum instantaneous reward, γ is as defined in Lemma 3 and $v_\mu^{m_t}(\pi, ax_{<t})$ is the value of policy π as defined in Definition 3.

Proof. First define $\rho(x_{i:j} \,|\, a_{1:j}, x_{<i}) := \rho(x_{1:j} \,|\, a_{1:j}) / \rho(x_{<i} \,|\, a_{<i})$ for $i < j$, for any environment model ρ and let $a_{t:m_t}$ be the actions chosen by π at times t to m_t. Now

$$\left| v_\Upsilon^{m_t}(\pi, ax_{<t}) - v_\mu^{m_t}(\pi, ax_{<t}) \right| \quad = \quad \left| \sum_{x_{t:m_t}} (r_t + \cdots + r_{m_t}) \left[\Upsilon(x_{t:m_t} \,|\, a_{1:m_t}, x_{<t}) - \mu(x_{t:m_t} \,|\, a_{1:m_t}, x_{<t}) \right] \right|$$

$$\begin{aligned}
&\leq \sum_{x_{t:m_t}} (r_t + \cdots + r_{m_t}) \left| \Upsilon(x_{t:m_t} \,|\, a_{1:m_t}, x_{<t}) - \mu(x_{t:m_t} \,|\, a_{1:m_t}, x_{<t}) \right| \\
&\leq m_t r_{max} \sum_{x_{t:m_t}} \left| \Upsilon(x_{t:m_t} \,|\, a_{1:m_t}, x_{<t}) - \mu(x_{t:m_t} \,|\, a_{1:m_t}, x_{<t}) \right| \\
&=: m_t r_{max} A_{t:m_t}(\mu \parallel \Upsilon).
\end{aligned}$$

Applying this bound, a property of absolute distance (Hutter, 2005, Lemma 3.11) and the chain rule for KL-divergence (Cover & Thomas, 1991, p. 24) gives

$$\begin{aligned}
\sum_{t=1}^{b} \mathbb{E}_{x_{<t}\sim\mu} \left[\left(v_{\Upsilon}^{m_t}(\pi, ax_{<t}) - v_{\mu}^{m_t}(\pi, ax_{<t}) \right)^2 \right] &\leq m_t^2 r_{max}^2 \sum_{t=1}^{b} \mathbb{E}_{x_{<t}\sim\mu} \left[A_{t:m_t}(\mu \parallel \Upsilon)^2 \right] \\
\leq 2H^2 r_{max}^2 \sum_{t=1}^{b} \mathbb{E}_{x_{<t}\sim\mu} \left[D_{t:m_t}(\mu \parallel \Upsilon) \right] &= 2H^2 r_{max}^2 \sum_{t=1}^{b} \sum_{i=t}^{m_t} \mathbb{E}_{x_{<i}\sim\mu} \left[D_{i:i}(\mu \parallel \Upsilon) \right] \\
\leq 2H^3 r_{max}^2 \sum_{t=1}^{b} \mathbb{E}_{x_{<t}\sim\mu} \left[D_{t:t}(\mu \parallel \Upsilon) \right] &= 2H^3 r_{max}^2 D_{1:b}(\mu \parallel \Upsilon),
\end{aligned}$$

where $D_{i:j}(\mu \parallel \Upsilon) := \sum_{x_{i:j}} \mu(x_{i:j} \,|\, a_{1:j}, x_{<i}) \ln(\Upsilon(x_{i:j} \,|\, a_{1:j}, x_{<i}) / \mu(x_{i:j} \,|\, a_{1:j}, x_{<i}))$. The final inequality uses the fact that any particular $D_{i:i}(\mu \parallel \Upsilon)$ term appears at most H times in the preceding double sum. Now define $\rho_M(\cdot \,|\, a_{1:b}) := \Pr(\cdot \,|\, (M_1, \ldots, M_k), a_{1:b})$ and we have

$$\begin{aligned}
D_{1:b}(\mu \parallel \Upsilon) &= \sum_{x_{1:b}} \mu(x_{1:b} \,|\, a_{1:b}) \ln \left[\frac{\mu(x_{1:b} \,|\, a_{1:b})}{\rho_M(x_{1:b} \,|\, a_{1:b})} \frac{\rho_M(x_{1:b} \,|\, a_{1:b})}{\Upsilon(x_{1:b} \,|\, a_{1:b})} \right] \\
&= \sum_{x_{1:b}} \mu(x_{1:b} \,|\, a_{1:b}) \ln \frac{\mu(x_{1:b} \,|\, a_{1:b})}{\rho_M(x_{1:b} \,|\, a_{1:b})} + \sum_{x_{1:b}} \mu(x_{1:b} \,|\, a_{1:b}) \ln \frac{\rho_M(x_{1:b} \,|\, a_{1:b})}{\Upsilon(x_{1:b} \,|\, a_{1:b})} \\
&\leq D_{1:b}(\mu \,\|\, \rho_M) + \sum_{x_{1:b}} \mu(x_{1:b} \,|\, a_{1:b}) \ln \frac{\rho_M(x_{1:b} \,|\, a_{1:b})}{w_0^{\rho_M} \rho_M(x_{1:b} \,|\, a_{1:b})} \\
&= D_{1:b}(\mu \,\|\, \rho_M) + \sum_{i=1}^{k} \Gamma_{D_i}(M_i)
\end{aligned}$$

where $w_0^{\rho_M} := 2^{-\sum_{i=1}^{k} \Gamma_{D_i}(M_i)}$ and the final inequality follows by dropping all but ρ_M's contribution to Equation (38). Using Lemma 3 to bound $D_{1:b}(\mu \,\|\, \rho_M)$ now gives the desired result. □

For any fixed H, Theorem 2 shows that the cumulative expected squared difference of the true and Υ values is bounded by a term that grows at the rate of $O(\log b)$. The average expected squared difference of the two values then goes down to zero at the rate of $O(\frac{\log b}{b})$. This implies that for sufficiently large b, the value estimates using Υ in place of μ converge for any fixed policy π. Importantly, this includes the fixed horizon expectimax policy with respect to Υ.

6.2 Convergence to Optimal Policy

This section presents a result for n-Markov environments that are both ergodic and stationary. Intuitively, this class of environments never allow the agent to make a mistake from which it can no longer recover. Thus in these environments an agent that learns from its mistakes can hope to achieve a long-term average reward that will approach optimality.

Definition 10. *An n-Markov environment μ is said to be ergodic if there exists a policy π such that every sub-history $s \in (\mathcal{A} \times \mathcal{X})^n$ possible in μ occurs infinitely often (with probability* 1*) in the history generated by an agent/environment pair (π, μ).*

Definition 11. *A sequence of policies $\{\pi_1, \pi_2, \dots\}$ is said to be self optimising with respect to model class $\mathcal{M}$ if*

$$\frac{1}{m} v_\rho^m(\pi_m, \epsilon) - \frac{1}{m} V_\rho^m(\epsilon) \to 0 \qquad \textit{as} \quad m \to \infty \quad \textit{for all} \quad \rho \in \mathcal{M}. \tag{41}$$

A self optimising policy has the same long-term average expected future reward as the optimal policy for any environment in $\mathcal{M}$. In general, such policies cannot exist for all model classes. We restrict our attention to the set of stationary, ergodic n-Markov environments since these are what can be modeled effectively by FAC-CTW. The ergodicity property ensures that no possible percepts are precluded due to earlier actions by the agent. The stationarity property ensures that the environment is sufficiently well behaved for a PST to learn a fixed set of parameters.

We now prove a lemma in preparation for our main result.

Lemma 4. *Any stationary, ergodic n-Markov environment can be modeled by a finite, ergodic MDP.*

Proof. Given an ergodic n-Markov environment μ, with associated action space $\mathcal{A}$ and percept space $\mathcal{X}$, an equivalent, finite MDP (S, A, T, R) can be constructed from μ by defining the state space as $S := (\mathcal{A} \times \mathcal{X})^n$, the action space as $A := \mathcal{A}$, the transition probability as $T_a(s, s') := \mu(o'r' \mid hsa)$ and the reward function as $R_a(s, s') := r'$, where s' is the suffix formed by deleting the leftmost action/percept pair from $sao'r'$ and h is an arbitrary history from $(\mathcal{A} \times \mathcal{X})^*$. $T_a(s, s')$ is well defined for arbitrary h since μ is stationary, therefore Eq. (36) applies. Definition 10 implies that the derived MDP is ergodic. □

Theorem 3. *Given a mixture environment model ξ over a model class $\mathcal{M}$ consisting of a countable set of stationary, ergodic n-Markov environments, the sequence of policies $\left\{\pi_1^\xi, \pi_2^\xi, \dots\right\}$ where*

$$\pi_b^\xi(ax_{<t}) := \arg\max_{a_t \in \mathcal{A}} V_\xi^{b-t+1}(ax_{<t}a_t) \tag{42}$$

for $1 \le t \le b$, is self-optimising with respect to model class $\mathcal{M}$.

Proof. By applying Lemma 4 to each $\rho \in \mathcal{M}$, an equivalent model class $\mathcal{N}$ of finite, ergodic MDPs can be produced. We know from Hutter (2005, Thm. 5.38) that a sequence of policies for $\mathcal{N}$ that is self-optimising exists. This implies the existence of a corresponding sequence of policies for $\mathcal{M}$ that is self-optimising. Using the work of Hutter (2005, Thm. 5.29), this implies that the sequence of policies $\left\{\pi_1^\xi, \pi_2^\xi, \dots\right\}$ is self optimising. □

Theorem 3 says that by choosing a sufficiently large lifespan b, the average reward for an agent following policy π_b^ξ can be made arbitrarily close to the optimal average reward with respect to the true environment.

Theorem 3 and the consistency of the ρUCT algorithm (17) give support to the claim that the MC-AIXI(fac-ctw) agent is self-optimising with respect to the class of stationary, ergodic, n-Markov environments. The argument isn't completely rigorous, since the usage of the KT-estimator implies that the model class of FAC-CTW contains an uncountable number of models. Our conclusion is not entirely unreasonable however. The justification is that a countable mixture of PSTs behaving similarly to the FAC-CTW mixture can be formed by replacing each PST leaf node KT-estimator with a finely grained, discrete Bayesian mixture predictor. Under this interpretation, a floating point implementation of the KT-estimator would correspond to a computationally feasible approximation of the above.

The results used in the proof of Theorem 3 can be found in the works of Hutter (2002b) and Legg and Hutter (2004). An interesting area for future research would be to investigate whether a self-optimising result similar to the work of Hutter (2005, Thm. 5.29) holds for continuous mixtures.

6.3 Computational Properties

The FAC-CTW algorithm grows each context tree data structure dynamically. With a context depth D, there are at most $O(tD \log(|\mathcal{O}||\mathcal{R}|))$ nodes in the set of context trees after t cycles. In practice, this is considerably less than $\log(|\mathcal{O}||\mathcal{R}|)2^D$, which is the number of nodes in a fully grown set of context trees. The time complexity of FAC-CTW is also impressive; $O(Dm \log(|\mathcal{O}||\mathcal{R}|))$ to generate the m percepts needed to perform a single ρUCT simulation and $O(D \log(|\mathcal{O}||\mathcal{R}|))$ to process each new piece of experience. Importantly, these quantities are not dependent on t, which means that the performance of our agent does not degrade with time. Thus it is reasonable to run our agent in an online setting for millions of cycles. Furthermore, as FAC-CTW is an exact algorithm, we do not suffer from approximation issues that plague sample based approaches to Bayesian learning.

6.4 Efficient Combination of FAC-CTW with ρUCT

Earlier, we showed how FAC-CTW can be used in an online setting. An additional property however is needed for efficient use within ρUCT. Before SAMPLE is invoked, FAC-CTW will have computed a set of context trees for a history of length t. After a complete trajectory is sampled, FAC-CTW will now contain a set of context trees for a history of length $t + m$. The original set of context trees now needs to be restored. Saving and copying the original context trees is unsatisfactory, as is rebuilding them from scratch in $O(tD \log(|\mathcal{O}||\mathcal{R}|))$ time. Luckily, the original set of context trees can be recovered efficiently by traversing the history at time $t + m$ in reverse, and performing an inverse update operation on each of the D affected nodes in the relevant context tree, for each bit in the sample trajectory. This takes $O(Dm \log(|\mathcal{O}||\mathcal{R}|))$ time. Alternatively, a copy on write implementation can be used to modify the context trees during the simulation phase, with the modified copies of each context node discarded before SAMPLE is invoked again.

6.5 Exploration/Exploitation in Practice

Bayesian belief updating combines well with expectimax based planning. Agents using this combination, such as AIXI and MC-AIXI(FAC-CTW), will automatically perform information gathering actions if the expected reduction in uncertainty would lead to higher expected future reward. Since AIXI is a mathematical notion, it can simply take a large initial planning horizon b, e.g. its maximal lifespan, and then at each cycle t choose greedily with respect to Equation (1) using a *remaining horizon* of $b - t + 1$. Unfortunately in the case of MC-AIXI(FAC-CTW), the situation is complicated by issues of limited computation.

In theory, the MC-AIXI(FAC-CTW) agent could always perform the action recommended by ρUCT. In practice however, performing an expectimax operation with a remaining horizon of $b-t+1$ is not feasible, even using Monte-Carlo approximation. Instead we use as large a fixed search hori-

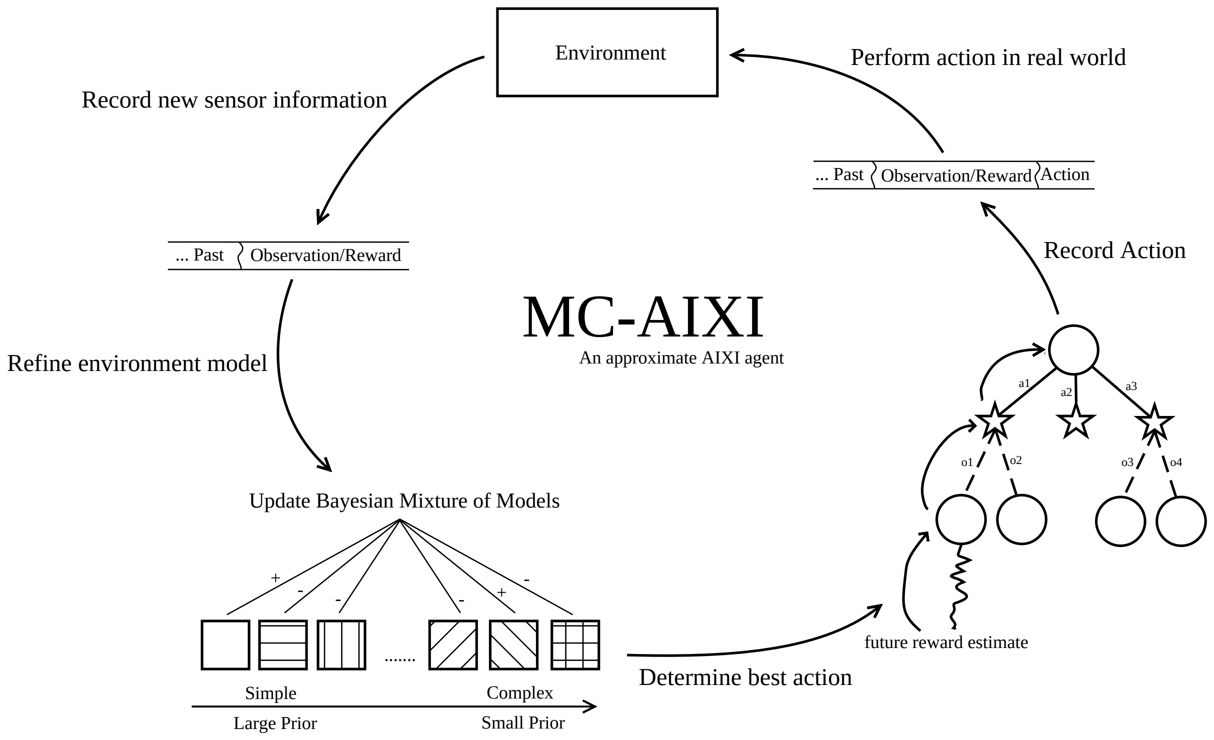
Environment
Perform action in real world
Record new sensor information
... Past Observation/Reward Action
... Past Observation/Reward
Record Action
MC-AIXI
An approximate AIXI agent
Refine environment model
a1
a2
a3
o1
o2
o3
o4
Update Bayesian Mixture of Models
+
-
-
-
+
-
....
future reward estimate
Determine best action
Simple
Complex
Large Prior
Small Prior

Figure 4: The MC-AIXI agent loop

zon as we can afford computationally, and occasionally force exploration according to some heuristic policy. The intuition behind this choice is that in many domains, good behaviour can be achieved by using a small amount of planning if the dynamics of the domain are known. Note that it is still possible for ρUCT to recommend an exploratory action, but only if the benefits of this information can be realised within its limited planning horizon. Thus, a limited amount of exploration can help the agent avoid local optima with respect to its present set of beliefs about the underlying environment. Other online reinforcement learning algorithms such as SARSA(λ) (Sutton & Barto, 1998), U-Tree (McCallum, 1996) or Active-LZ (Farias, Moallemi, Van Roy, & Weissman, 2010) employ similar such strategies.

6.6 Top-level Algorithm

At each time step, MC-AIXI(fac-ctw) first invokes the ρUCT routine with a fixed horizon to estimate the value of each candidate action. An action is then chosen according to some policy that balances exploration with exploitation, such as ϵ-Greedy or Softmax (Sutton & Barto, 1998). This action is communicated to the environment, which responds with an observation-reward pair. The agent then incorporates this information into Υ using the FAC-CTW algorithm and the cycle repeats. Figure 4 gives an overview of the agent/environment interaction loop.

Domain	$\|\mathcal{A}\|$	$\|\mathcal{O}\|$	Aliasing	Noisy $\mathcal{O}$	Uninformative $\mathcal{O}$
1d-maze	2	1	yes	no	yes
Cheese Maze	4	16	yes	no	no
Tiger	3	3	yes	yes	no
Extended Tiger	4	3	yes	yes	no
4×4 Grid	4	1	yes	no	yes
TicTacToe	9	19683	no	no	no
Biased Rock-Paper-Scissor	3	3	no	yes	no
Kuhn Poker	2	6	yes	yes	no
Partially Observable Pacman	4	2^{16}	yes	no	no

Table 1: Domain characteristics

7. Experimental Results

We now measure our agent's performance across a number of different domains. In particular, we focused on learning and solving some well-known benchmark problems from the POMDP literature. Given the full POMDP model, computation of the optimal policy for each of these POMDPs is not difficult. However, our requirement of having to both learn a model of the environment, as well as find a good policy online, *significantly* increases the difficulty of these problems. From the agent's perspective, our domains contain perceptual aliasing, noise, partial information, and inherent stochastic elements.

7.1 Domains

Our test domains are now described. Their characteristics are summarized in Table 1.

1d-maze. The 1d-maze is a simple problem from the work of Cassandra, Kaelbling, and Littman (1994). The agent begins at a random, non-goal location within a 1×4 maze. There is a choice of two actions: left or right. Each action transfers the agent to the adjacent cell if it exists, otherwise it has no effect. If the agent reaches the third cell from the left, it receives a reward of 1. Otherwise it receives a reward of 0. The distinguishing feature of this problem is that the observations are *uninformative*; every observation is the same regardless of the agent's actual location.

Cheese Maze. This well known problem is due to McCallum (1996). The agent is a mouse inside a two dimensional maze seeking a piece of cheese. The agent has to choose one of four actions: move up, down, left or right. If the agent bumps into a wall, it receives a penalty of -10. If the agent finds the cheese, it receives a reward of 10. Each movement into a free cell gives a penalty of -1. The problem is depicted graphically in Figure 5. The number in each cell represents the decimal equivalent of the four bit binary observation (0 for a free neighbouring cell, 1 for a wall) the mouse receives in each cell. The problem exhibits perceptual aliasing in that a single observation is potentially ambiguous.

Tiger. This is another familiar domain from the work of Kaelbling, Littman, and Cassandra (1995). The environment dynamics are as follows: a tiger and a pot of gold are hidden behind one of two doors. Initially the agent starts facing both doors. The agent has a choice of one of three actions: listen, open the left door, or open the right door. If the agent opens the door hiding the

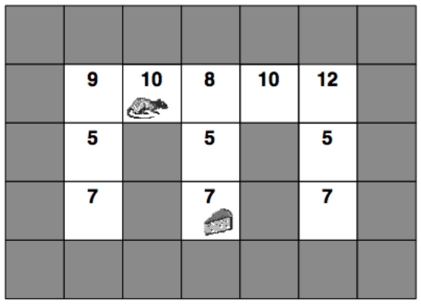

Figure 5: The cheese maze

tiger, it suffers a -100 penalty. If it opens the door with the pot of gold, it receives a reward of 10. If the agent performs the listen action, it receives a penalty of -1 and an observation that correctly describes where the tiger is with 0.85 probability.

Extended Tiger. The problem setting is similar to Tiger, except that now the agent begins sitting down on a chair. The actions available to the agent are: stand, listen, open the left door, and open the right door. Before an agent can successfully open one of the two doors, it must stand up. However, the listen action only provides information about the tiger's whereabouts when the agent is sitting down. Thus it is necessary for the agent to plan a more intricate series of actions before it sees the optimal solution. The reward structure is slightly modified from the simple Tiger problem, as now the agent gets a reward of 30 when finding the pot of gold.

4×4 Grid. The agent is restricted to a 4×4 grid world. It can move either up, down, right or left. If the agent moves into the bottom right corner, it receives a reward of 1, and it is randomly teleported to one of the remaining 15 cells. If it moves into any cell other than the bottom right corner cell, it receives a reward of 0. If the agent attempts to move into a non-existent cell, it remains in the same location. Like the 1d-maze, this problem is also uninformative but on a much larger scale. Although this domain is simple, it does require some subtlety on the part of the agent. The correct action depends on what the agent has tried before at previous time steps. For example, if the agent has repeatedly moved right and not received a positive reward, then the chances of it receiving a positive reward by moving down are increased.

TicTacToe. In this domain, the agent plays repeated games of TicTacToe against an opponent who moves randomly. If the agent wins the game, it receives a reward of 2. If there is a draw, the agent receives a reward of 1. A loss penalises the agent by -2. If the agent makes an illegal move, by moving on top of an already filled square, then it receives a reward of -3. A legal move that does not end the game earns no reward.

Biased Rock-Paper-Scissors. This domain is taken from the work of Farias et al. (2010). The agent repeatedly plays Rock-Paper-Scissor against an opponent that has a slight, predictable bias in its strategy. If the opponent has won a round by playing rock on the previous cycle, it will always play rock at the next cycle; otherwise it will pick an action uniformly at random. The agent's observation is the most recently chosen action of the opponent. It receives a reward of 1 for a win, 0 for a draw and -1 for a loss.

Kuhn Poker. Our next domain involves playing Kuhn Poker (Kuhn, 1950; Hoehn, Southey, Holte, & Bulitko, 2005) against an opponent playing a Nash strategy. Kuhn Poker is a simplified, zero-sum, two player poker variant that uses a deck of three cards: a King, Queen and Jack. Whilst considerably less sophisticated than popular poker variants such as Texas Hold'em, well-known strategic concepts such as bluffing and slow-playing remain characteristic of strong play.

In our setup, the agent acts second in a series of rounds. Two actions, pass or bet, are available to each player. A bet action requires the player to put an extra chip into play. At the beginning of each round, each player puts a chip into play. The opponent then decides whether to pass or bet; betting will win the round if the agent subsequently passes, otherwise a showdown will occur. In a showdown, the player with the highest card wins the round. If the opponent passes, the agent can either bet or pass; passing leads immediately to a showdown, whilst betting requires the opponent to either bet to force a showdown, or to pass and let the agent win the round uncontested. The winner of the round gains a reward equal to the total chips in play, the loser receives a penalty equal to the number of chips they put into play this round. At the end of the round, all chips are removed from play and another round begins.

Kuhn Poker has a known optimal solution. Against a first player playing a Nash strategy, the second player can obtain at most an average reward of $\frac{1}{18}$ per round.

Partially Observable Pacman. This domain is a partially observable version of the classic Pacman game. The agent must navigate a 17×17 maze and eat the pills that are distributed across the maze. Four ghosts roam the maze. They move initially at random, until there is a Manhattan distance of 5 between them and Pacman, whereupon they will aggressively pursue Pacman for a short duration. The maze structure and game are the same as the original arcade game, however the Pacman agent is hampered by partial observability. Pacman is unaware of the maze structure and only receives a 4-bit observation describing the wall configuration at its current location. It also does not know the exact location of the ghosts, receiving only 4-bit observations indicating whether a ghost is visible (via direct line of sight) in each of the four cardinal directions. In addition, the locations of the food pellets are unknown except for a 3-bit observation that indicates whether food can be smelt within a Manhattan distance of 2, 3 or 4 from Pacman's location, and another 4-bit observation indicating whether there is food in its direct line of sight. A final single bit indicates whether Pacman is under the effects of a power pill. At the start of each episode, a food pellet is placed down with probability 0.5 at every empty location on the grid. The agent receives a penalty of 1 for each movement action, a penalty of 10 for running into a wall, a reward of 10 for each food pellet eaten, a penalty of 50 if it is caught by a ghost, and a reward of 100 for collecting all the food. If multiple such events occur, then the total reward is cumulative, i.e. running into a wall and being caught would give a penalty of 60. The episode resets if the agent is caught or if it collects all the food.

Figure 6 shows a graphical representation of the partially observable Pacman domain. This problem is the largest domain we consider, with an unknown optimal policy. The main purpose of this domain is to show the scaling properties of our agent on a challenging problem. Note that this domain is fundamentally different to the Pacman domain used in (Silver & Veness, 2010). In addition to using a different observation space, we also do not assume that the true environment is known a-priori.

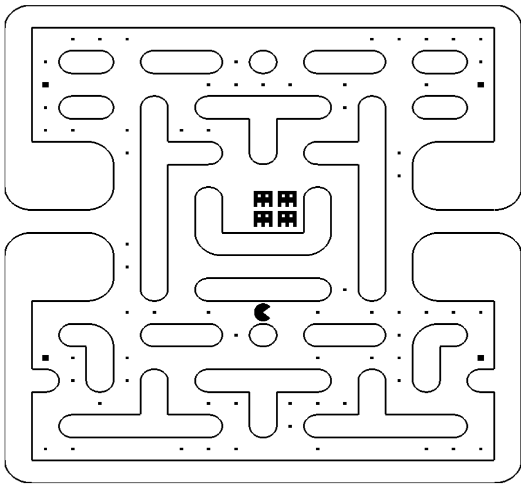

Figure 6: A screenshot (converted to black and white) of the PacMan domain

7.2 Experimental Setup

We now evaluate the performance of the MC-AIXI(fac-ctw) agent. To help put our results into perspective, we implemented and directly compared against two competing algorithms from the model-based general reinforcement learning literature: U-Tree (McCallum, 1996) and Active-LZ (Farias et al., 2010). The two algorithms are described on page 133 in Section 8. As FAC-CTW subsumes Action Conditional CTW, we do not evaluate it in this paper; results using Action Conditional CTW can be found in our previous work (Veness et al., 2010). The performance of the agent using FAC-CTW is no worse and in some cases slightly better than the previous results.

Each agent communicates with the environment over a binary channel. A cycle begins with the agent sending an action a to the environment, which then responds with a percept x. This cycle is then repeated. A fixed number of bits are used to encode the action, observation and reward spaces for each domain. These are specified in Table 2. No constraint is placed on how the agent interprets the observation component; e.g., this could be done at either the bit or symbol level. The rewards are encoded naively, i.e. the bits corresponding to the reward are interpreted as unsigned integers. Negative rewards are handled (without loss of generality) by offsetting all of the rewards so that they are guaranteed to be non-negative. These offsets are removed from the reported results.

The process of gathering results for each of the three agents is broken into two phases: model learning and model evaluation. The model learning phase involves running each agent with an exploratory policy to build a model of the environment. This learnt model is then evaluated at various points in time by running the agent without exploration for 5000 cycles and reporting the average reward per cycle. More precisely, at time t the average reward per cycle is defined as $\frac{1}{5000}\sum_{i=t+1}^{t+5000} r_i$, where r_i is the reward received at cycle i. Having two separate phases reduces the influence of the agent's earlier exploratory actions on the reported performance. All of our experiments were performed on a dual quad-core Intel 2.53Ghz Xeon with 24 gigabytes of memory.

Table 3 outlines the parameters used by MC-AIXI(fac-ctw) during the model learning phase. The context depth parameter D specifies the maximal number of recent bits used by FAC-CTW. The ρUCT search horizon is specified by the parameter m. Larger D and m increase the capabilities of our agent, at the expense of linearly increasing computation time; our values represent

Domain	$\mathcal{A}$ bits	$\mathcal{O}$ bits	$\mathcal{R}$ bits
1d-maze	1	1	1
Cheese Maze	2	4	5
Tiger	2	2	7
Extended Tiger	2	3	8
4×4 Grid	2	1	1
TicTacToe	4	18	3
Biased Rock-Paper-Scissor	2	2	2
Kuhn Poker	1	4	3
Partially Observable Pacman	2	16	8

Table 2: Binary encoding of the domains

Domain	D	m	ϵ	γ	ρUCT Simulations
1d-maze	32	10	0.9	0.99	500
Cheese Maze	96	8	0.999	0.9999	500
Tiger	96	5	0.99	0.9999	500
Extended Tiger	96	4	0.99	0.99999	500
4×4 Grid	96	12	0.9	0.9999	500
TicTacToe	64	9	0.9999	0.999999	500
Biased Rock-Paper-Scissor	32	4	0.999	0.99999	500
Kuhn Poker	42	2	0.99	0.9999	500
Partial Observable Pacman	96	4	0.9999	0.99999	500

Table 3: MC-AIXI(FAC-CTW) model learning configuration

an appropriate compromise between these two competing dimensions for each problem domain. Exploration during the model learning phase is controlled by the ϵ and γ parameters. At time t, MC-AIXI(FAC-CTW) explores a random action with probability $\gamma^t \epsilon$. During the model evaluation phase, exploration is disabled, with results being recorded for varying amounts of experience and search effort.

The Active-LZ algorithm is fully specified in the work of Farias et al. (2010). It contains only two parameters, a discount rate and a policy that balances between exploration and exploitation. During the model learning phase, a discount rate of 0.99 and ϵ-Greedy exploration (with $\epsilon = 0.95$) were used. Smaller exploration values (such as 0.05, 0.2, 0.5) were tried, as well as policies that decayed ϵ over time, but these surprisingly gave slightly worse performance during testing. As a sanity check, we confirmed that our implementation could reproduce the experimental results reported in the work of Farias et al. (2010). During the model evaluation phase, exploration is disabled.

The situation is somewhat more complicated for U-Tree, as it is more of a general agent framework than a completely specified algorithm. Due to the absence of a publicly available reference implementation, a number of implementation-specific decisions were made. These included the choice of splitting criteria, how far back in time these criteria could be applied, the frequency of

Domain	ϵ	Test Fringe	α
1d-maze	0.05	100	0.05
Cheese Maze	0.2	100	0.05
Tiger	0.1	100	0.05
Extended Tiger	0.05	200	0.01
4×4 Grid	0.05	100	0.05
TicTacToe	0.05	1000	0.01
Biased Rock-Paper-Scissor	0.05	100	0.05
Kuhn Poker	0.05	200	0.05

Table 4: U-Tree model learning configuration

fringe tests, the choice of p-value for the Kolmogorov-Smirnov test, the exploration/exploitation policy and the learning rate. The main design decisions are listed below:

- A split could be made on any action, or on the status of any single bit of an observation.
- The maximum number of steps backwards in time for which a utile distinction could be made was set to 5.
- The frequency of fringe tests was maximised given realistic resource constraints. Our choices allowed for 5×10^4 cycles of interaction to be completed on each domain within 2 days of training time.
- Splits were tried in order from the most temporally recent to the most temporally distant.
- ϵ-Greedy exploration strategy was used, with ϵ tuned separately for each domain.
- The learning rate α was tuned for each domain.

To help make the comparison as fair as possible, an effort was made to tune U-Tree's parameters for each domain. The final choices for the model learning phase are summarised in Table 4. During the model evaluation phase, both exploration and testing of the fringe are disabled.

Source Code. The code for our U-Tree, Active-LZ and MC-AIXI(fac-ctw) implementations can be found at: `http://jveness.info/software/mcaixi_jair_2010.zip`.

7.3 Results

Figure 7 presents our main set of results. Each graph shows the performance of each agent as it accumulates more experience. The performance of MC-AIXI(fac-ctw) matches or exceeds U-Tree and Active-LZ on all of our test domains. Active-LZ steadily improved with more experience, however it learnt significantly more slowly than both U-Tree and MC-AIXI(fac-ctw). U-Tree performed well in most domains, however the overhead of testing for splits limited its ability to be run for long periods of time. This is the reason why some data points for U-Tree are missing from the graphs in Figure 7. This highlights the advantage of algorithms that take constant time per cycle, such as MC-AIXI(fac-ctw) and Active-LZ. Constant time isn't enough however, especially when large observation spaces are involved. Active-LZ works at the symbol level, with the algorithm given by Farias et al. (2010) requiring an exhaustive enumeration of the percept space on each cycle. This is not possible in reasonable time for the larger TicTacToe domain, which is why no Active-LZ result

Domain	Experience	ρUCT Simulations	Search Time per Cycle
1d Maze	5×10^3	250	0.1s
Cheese Maze	2.5×10^3	500	0.5s
Tiger	2.5×10^4	25000	10.6s
Extended Tiger	5×10^4	25000	12.6s
4×4 Grid	2.5×10^4	500	0.3s
TicTacToe	5×10^5	2500	4.1s
Biased RPS	1×10^4	5000	2.5s
Kuhn Poker	5×10^6	250	0.1s

Table 5: Resources required for (near) optimal performance by MC-AIXI(FAC-CTW)

is presented. This illustrates an important advantage of MC-AIXI(FAC-CTW) and U-Tree, which have the ability to exploit structure *within* a single observation.

Figure 8 shows the performance of MC-AIXI(FAC-CTW) as the number of ρUCT simulations varies. The results for each domain were based on a model learnt from 5×10^4 cycles of experience, except in the case of TicTacToe where 5×10^5 cycles were used. So that results could be compared across domains, the average reward per cycle was normalised to the interval $[0, 1]$. As expected, domains that included a significant planning component (such as Tiger or Extended Tiger) required more search effort. Good performance on most domains was obtained using only 1000 simulations.

Given a sufficient number of ρUCT simulations and cycles of interaction, the performance of the MC-AIXI(FAC-CTW) agent approaches optimality on our test domains. The amount of resources needed for near optimal performance on each domain during the model evaluation phase is listed in Table 5. Search times are also reported. This shows that the MC-AIXI(FAC-CTW) agent can be realistically used on a present day workstation.

7.4 Discussion

The small state space induced by U-Tree has the benefit of limiting the number of parameters that need to be estimated from data. This can dramatically speed up the model-learning process. In contrast, both Active-LZ and our approach require a number of parameters proportional to the number of distinct contexts. This is one of the reasons why Active-LZ exhibits slow convergence in practice. This problem is much less pronounced in our approach for two reasons. First, the Ockham prior in CTW ensures that future predictions are dominated by PST structures that have seen enough data to be trustworthy. Secondly, value function estimation is decoupled from the process of context estimation. Thus it is reasonable to expect ρUCT to make good local decisions provided FAC-CTW can predict well. The downside however is that our approach requires search for action selection. Although ρUCT is an anytime algorithm, in practice more computation (at least on small domains) is required per cycle compared to approaches like Active-LZ and U-Tree that act greedily with respect to an estimated global value function.

The U-Tree algorithm is well motivated, but unlike Active-LZ and our approach, it lacks theoretical performance guarantees. It is possible for U-Tree to prematurely converge to a locally optimal state representation from which the heuristic splitting criterion can never recover. Furthermore, the splitting heuristic contains a number of configuration options that can dramatically influence its performance (McCallum, 1996). This parameter sensitivity somewhat limits the algorithm's

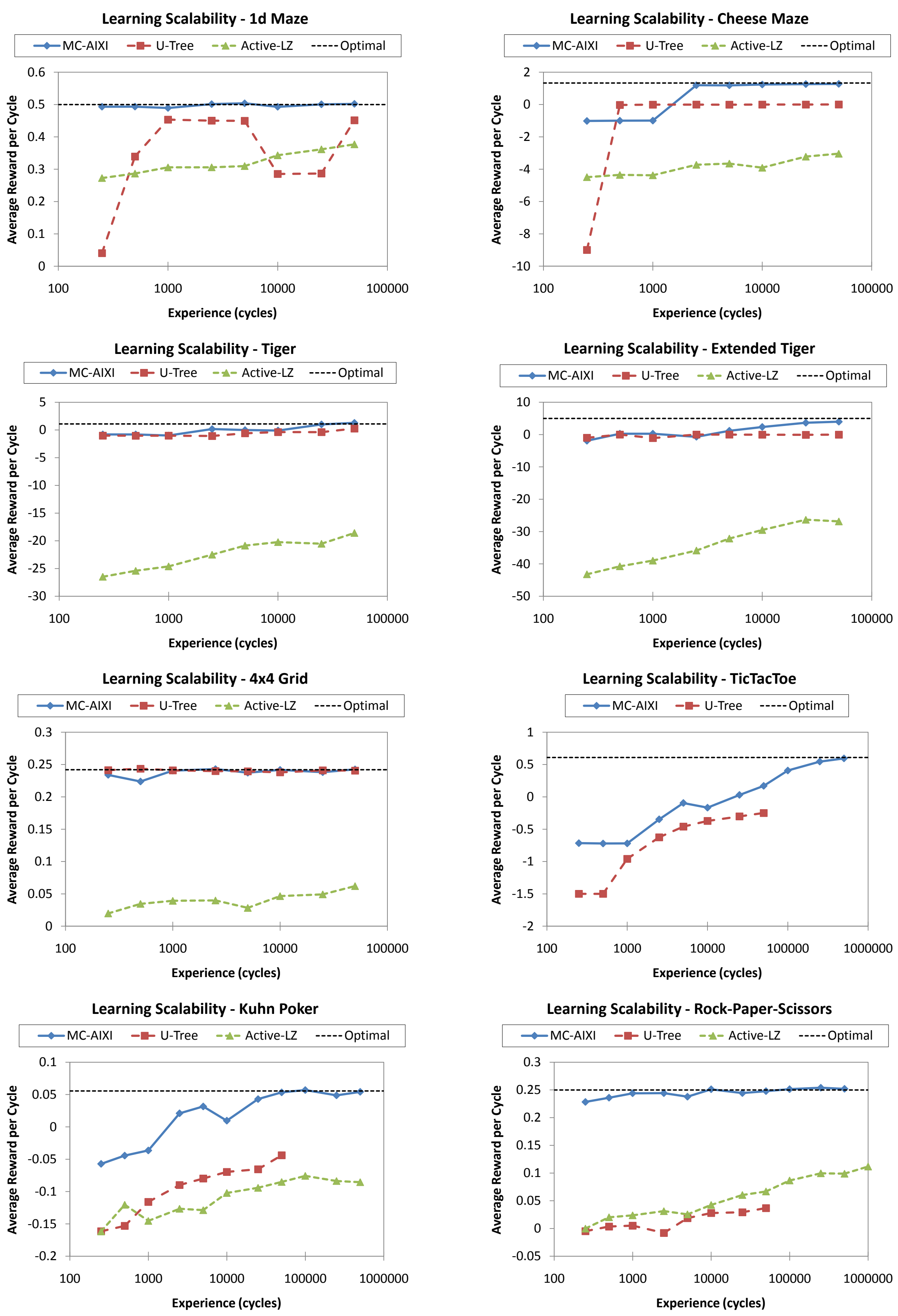

Figure 7: Average Reward per Cycle vs Experience

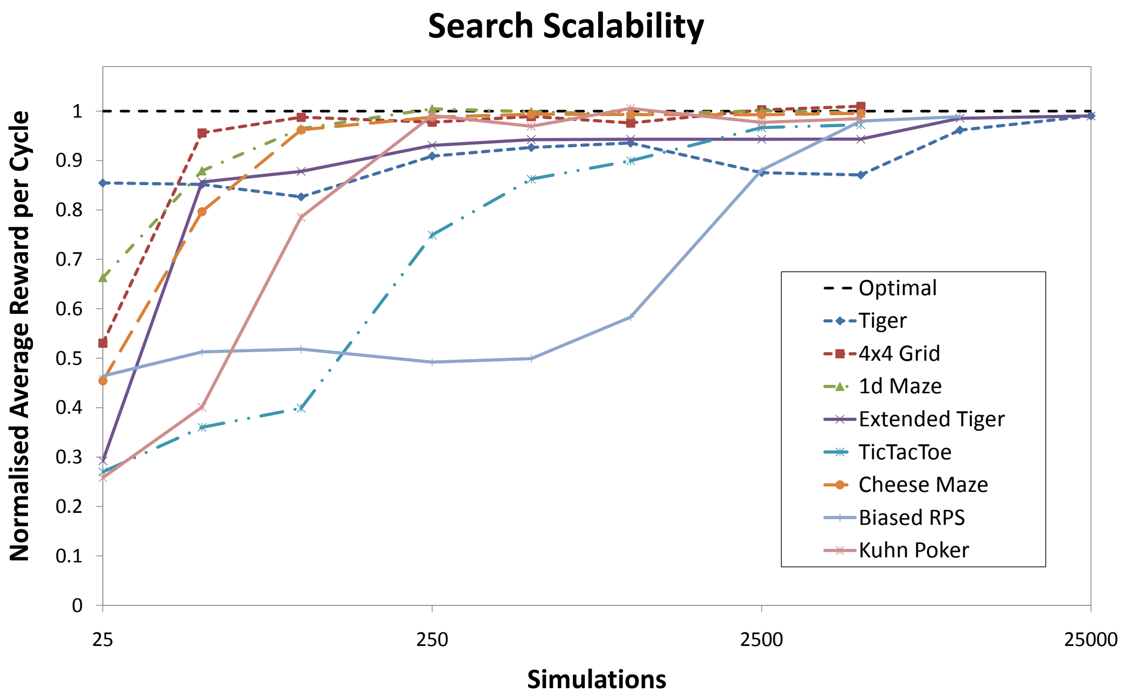

Figure 8: Performance versus ρUCT search effort

applicability to the general reinforcement learning problem. Still, our results suggest that further investigation of frameworks motivated along the same lines as U-Tree is warranted.

7.5 Comparison to 1-ply Rollout Planning

We now investigate the performance of ρUCT in comparison to an adaptation of the well-known 1-ply rollout-based planning technique of Bertsekas and Castanon (1999). In our setting, this works as follows: given a history h, an estimate $\hat{V}(ha)$ is constructed for each action $a \in \mathcal{A}$, by averaging the returns of many length m simulations initiated from ha. The first action of each simulation is sampled uniformly at random from $\mathcal{A}$, whilst the remaining actions are selected according to some heuristic rollout policy. Once a sufficient number of simulations have been completed, the action with the highest estimated value is selected. Unlike ρUCT, this procedure doesn't build a tree, nor is it guaranteed to converge to the depth m expectimax solution. In practice however, especially in noisy and highly stochastic domains, rollout-based planning can significantly improve the performance of an existing heuristic rollout policy (Bertsekas & Castanon, 1999).

Table 6 shows how the performance (given by average reward per cycle) differs when ρUCT is replaced by the 1-ply rollout planner. The amount of experience collected by the agent, as well as the total number of rollout simulations, is the same as in Table 5. Both ρUCT and the 1-ply planner use the same search horizon, heuristic rollout policy (each action is chosen uniformly at random) and total number of simulations for each decision. This is reasonable, since although ρUCT has a slightly higher overhead compared to the 1-ply rollout planner, this difference is negligible when taking into account the cost of simulating future trajectories using FAC-CTW. Also, similar to previous experiments, 5000 cycles of greedy action selection were used to evaluate the performance of the FAC-CTW + 1-ply rollout planning combination.

Domain	MC-AIXI(fac-ctw)	FAC-CTW + 1-ply MC
1d Maze	0.50	0.50
Cheese Maze	1.28	1.25
Tiger	1.12	1.11
Extended Tiger	**3.97**	**-0.97**
4x4 Grid	0.24	0.24
TicTacToe	0.60	0.59
Biased RPS	**0.25**	**0.20**
Kuhn Poker	0.06	0.06

Table 6: Average reward per cycle: ρUCT versus 1-ply rollout planning

Importantly, ρUCT never gives worse performance than the 1-ply rollout planner, and on some domains (shown in bold) performs better. The ρUCT algorithm provides a way of performing multi-step planning whilst retaining the considerable computational advantages of rollout based methods. In particular, ρUCT will be able to construct deep plans in regions of the search space where most of the probability mass is concentrated on a small set of the possible percepts. When such structure exists, ρUCT will automatically exploit it. In the worst case where the environment is highly noisy or stochastic, the performance will be similar to that of rollout based planning. Interestingly, on many domains the empirical performance of 1-ply rollout planning matched that of ρUCT. We believe this to be a byproduct of our modest set of test domains, where multi-step planning is less important than learning an accurate model of the environment.

7.6 Performance on a Challenging Domain

The performance of MC-AIXI(fac-ctw) was also evaluated on the challenging Partially Observable Pacman domain. This is an enormous problem. Even if the true environment were known, planning would still be difficult due to the 10^{60} distinct underlying states.

We first evaluated the performance of MC-AIXI(fac-ctw) online. A discounted ϵ-Greedy policy, which chose a random action at time t with probability $\epsilon\gamma^t$ was used. These parameters were instantiated with $\epsilon := 0.9999$ and $\gamma := 0.99999$. When not exploring, each action was determined by ρUCT using 500 simulations. Figure 10 shows both the average reward per cycle and the average reward across the most recent 5000 cycles.

The performance of this learnt model was then evaluated by performing 5000 steps of greedy action selection, at various time points, whilst varying the number of simulations used by ρUCT. Figure 9 shows obtained results. The agent's performance scales with both the number of cycles of interaction and the amount of search effort. The results in Figure 9 using 500 simulations are higher than in Figure 10 since the performance is no longer affected by the exploration policy or earlier behavior based on an inferior learnt model.

Visual inspection[1] of Pacman shows that the agent, whilst not playing perfectly, has already learnt a number of important concepts. It knows not to run into walls. It knows how to seek out food from the limited information provided by its sensors. It knows how to run away and avoid chasing ghosts. The main subtlety that it hasn't learnt yet is to aggressively chase down ghosts when it has eaten a red power pill. Also, its behaviour can sometimes become temporarily erratic

1. See `http://jveness.info/publications/pacman_jair_2010.wmv` for a graphical demonstration

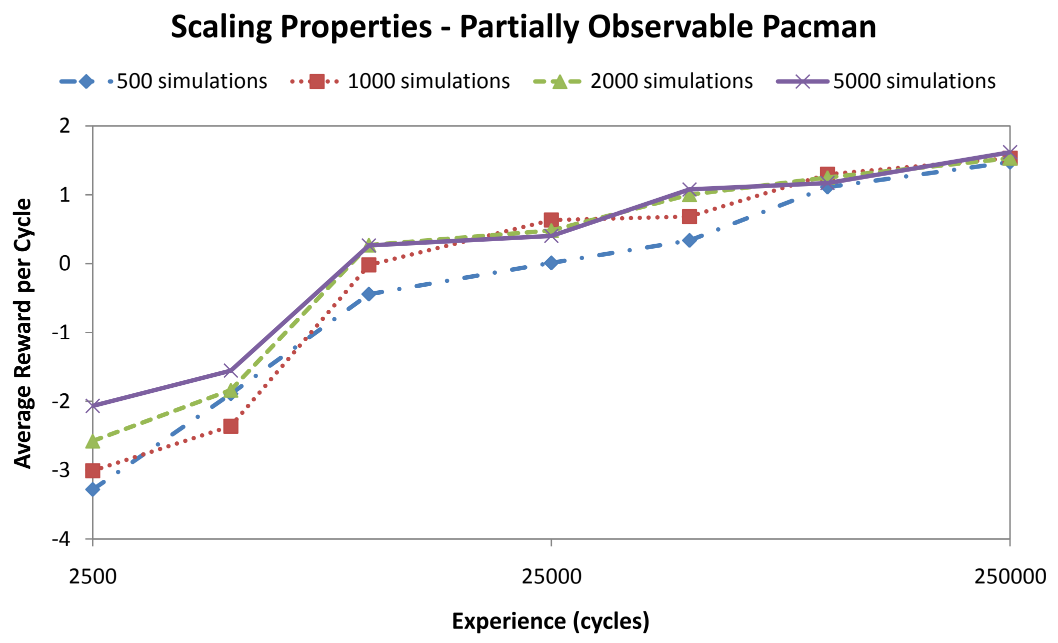

Figure 9: Scaling properties on a challenging domain

when stuck in a long corridor with no nearby food or visible ghosts. Still, the ability to perform reasonably on a large domain and exhibit consistent improvements makes us optimistic about the ability of the MC-AIXI(FAC-CTW) agent to scale with extra computational resources.

8. Discussion

We now discuss related work and some limitations of our current approach.

8.1 Related Work

There have been several attempts at studying the computational properties of AIXI. In the work of Hutter (2002a), an asymptotically optimal algorithm is proposed that, in parallel, picks and runs the fastest program from an enumeration of provably correct programs for any given well-defined problem. A similar construction that runs all programs of length less than l and time less than t per cycle and picks the best output (in the sense of maximising a provable lower bound for the true value) results in the optimal time bounded AIXItl agent (Hutter, 2005, Chp.7). Like Levin search (Levin, 1973), such algorithms are not practical in general but can in some cases be applied successfully (e.g., see Schmidhuber, 1997; Schmidhuber, Zhao, & Wiering, 1997; Schmidhuber, 2003, 2004). In tiny domains, universal learning is computationally feasible with brute-force search. In the work of Poland and Hutter (2006), the behaviour of AIXI is compared with a universal predicting-with-expert-advice algorithm (Poland & Hutter, 2005) in repeated 2×2 matrix games and is shown to exhibit different behaviour. A Monte-Carlo algorithm is proposed by Pankov (2008) that samples programs according to their algorithmic probability as a way of approximating Solomonoff's universal prior. A closely related algorithm is that of speed prior sampling (Schmidhuber, 2002).

We now move on to a discussion of the model-based general reinforcement learning literature. An early and influential work is the Utile Suffix Memory (USM) algorithm described by McCallum

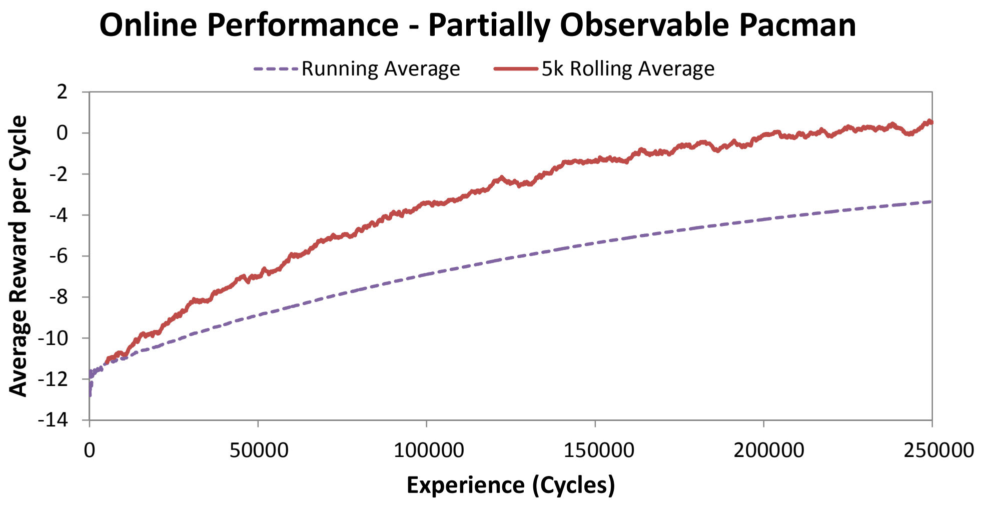

Figure 10: Online performance on a challenging domain

(1996). USM uses a suffix tree to partition the agent's history space into distinct states, one for each leaf in the suffix tree. Associated with each state/leaf is a Q-value, which is updated incrementally from experience like in Q-learning (Watkins & Dayan, 1992). The history-partitioning suffix tree is grown in an incremental fashion, starting from a single leaf node in the beginning. A leaf in the suffix tree is split when the history sequences that fall into the leaf are shown to exhibit statistically different Q-values. The USM algorithm works well for a number of tasks but could not deal effectively with noisy environments. Several extensions of USM to deal with noisy environments are investigated in the work of Shani and Brafman (2004) and Shani (2007).

U-Tree (McCallum, 1996) is an online agent algorithm that attempts to discover a compact state representation from a raw stream of experience. The main difference between U-Tree and USM is that U-Tree can discriminate between individual components within an observation. This allows U-Tree to more effectively handle larger observation spaces and ignore potentially irrelevant components of the observation vector. Each state is represented as the leaf of a suffix tree that maps history sequences to states. As more experience is gathered, the state representation is refined according to a heuristic built around the Kolmogorov-Smirnov test. This heuristic tries to limit the growth of the suffix tree to places that would allow for better prediction of future reward. Value Iteration is used at each time step to update the value function for the learnt state representation, which is then used by the agent for action selection.

Active-LZ (Farias et al., 2010) combines a Lempel-Ziv based prediction scheme with dynamic programming for control to produce an agent that is provably asymptotically optimal if the environment is n-Markov. The algorithm builds a context tree (distinct from the context tree built by CTW), with each node containing accumulated transition statistics and a value function estimate. These estimates are refined over time, allowing for the Active-LZ agent to steadily increase its performance. In Section 7, we showed that our agent compared favourably to Active-LZ.

The BLHT algorithm (Suematsu, Hayashi, & Li, 1997; Suematsu & Hayashi, 1999) uses symbol level PSTs for learning and an (unspecified) dynamic programming based algorithm for control. BLHT uses the most probable model for prediction, whereas we use a mixture model, which admits a much stronger convergence result. A further distinction is our usage of an Ockham prior instead of a uniform prior over PST models.

Predictive state representations (PSRs) (Littman, Sutton, & Singh, 2002; Singh, James, & Rudary, 2004; Rosencrantz, Gordon, & Thrun, 2004) maintain predictions of future experience. Formally, a PSR is a probability distribution over the agent's future experience, given its past experience. A subset of these predictions, the core tests, provide a sufficient statistic for all future experience. PSRs provide a Markov state representation, can represent and track the agent's state in partially observable environments, and provide a complete model of the world's dynamics. Unfortunately, exact representations of state are impractical in large domains, and some form of approximation is typically required. Topics such as improved learning or discovery algorithms for PSRs are currently active areas of research. The recent results of Boots, Siddiqi, and Gordon (2010) appear particularly promising.

Temporal-difference networks (Sutton & Tanner, 2004) are a form of predictive state representation in which the agent's state is approximated by abstract predictions. These can be predictions about future observations, but also predictions about future predictions. This set of interconnected predictions is known as the *question network*. Temporal-difference networks learn an approximate model of the world's dynamics: given the current predictions, the agent's action, and an observation vector, they provide new predictions for the next time-step. The parameters of the model, known as the *answer network*, are updated after each time-step by temporal-difference learning. Some promising recent results applying TD-Networks for prediction (but not control) to small POMDPs are given in (Makino, 2009).

In model-based Bayesian Reinforcement Learning (Strens, 2000; Poupart, Vlassis, Hoey, & Regan, 2006; Ross, Chaib-draa, & Pineau, 2008; Poupart & Vlassis, 2008), a distribution over (PO)MDP parameters is maintained. In contrast, we maintain an exact Bayesian mixture of PSTs, which are variable-order Markov models. The ρUCT algorithm shares similarities with Bayesian Sparse Sampling (Wang, Lizotte, Bowling, & Schuurmans, 2005). The main differences are estimating the leaf node values with a rollout function and using the UCB policy to direct the search.

8.2 Limitations

Our current AIXI approximation has two main limitations.

The first limitation is the restricted model class used for learning and prediction. Our agent will perform poorly if the underlying environment cannot be predicted well by a PST of bounded depth. Prohibitive amounts of experience will be required if a large PST model is needed for accurate prediction. For example, it would be unrealistic to think that our current AIXI approximation could cope with real-world image or audio data.

The second limitation is that unless the planning horizon is unrealistically small, our full Bayesian solution (using ρUCT and a mixture environment model) to the exploration/exploitation dilemma is computationally intractable. This is why our agent needs to be augmented by a heuristic exploration/exploitation policy in practice. Although this did not prevent our agent from obtaining optimal performance on our test domains, a better solution may be required for more challenging problems. In the MDP setting, considerable progress has been made towards resolving the exploration/exploitation issue. In particular, powerful PAC-MDP approaches exist for both model-based and model-free reinforcement learning agents (Brafman & Tennenholtz, 2003; Strehl, Li, Wiewiora,

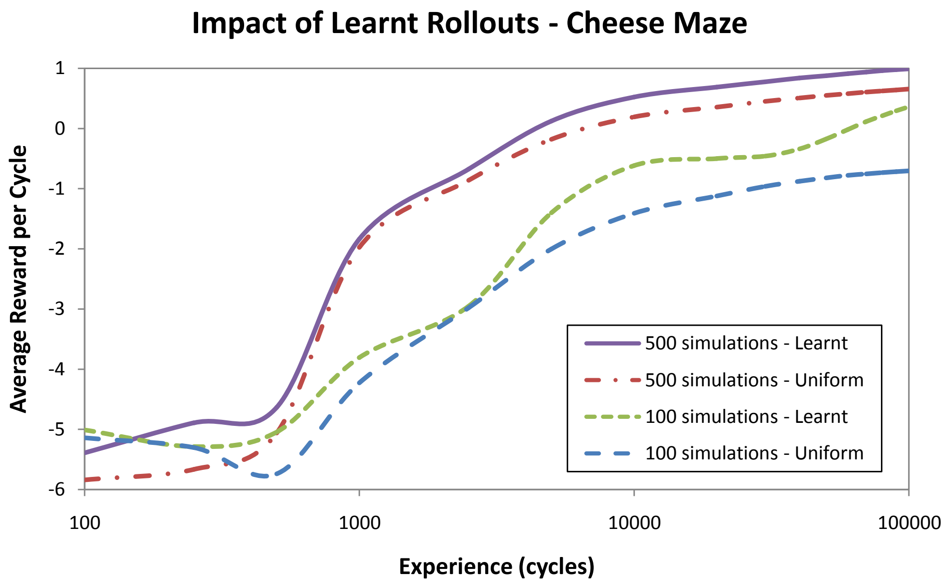

Figure 11: Online performance when using a learnt rollout policy on the Cheese Maze

Langford, & Littman, 2006; Strehl, Li, & Littman, 2009). It remains to be seen whether similar such principled approaches exist for history-based Bayesian agents.

9. Future Scalability

We now list some ideas that make us optimistic about the future scalability of our approach.

9.1 Online Learning of Rollout Policies for ρUCT

An important parameter to ρUCT is the choice of rollout policy. In MCTS methods for Computer Go, it is well known that search performance can be improved by using knowledge-based rollout policies (Gelly, Wang, Munos, & Teytaud, 2006). In the general agent setting, it would thus be desirable to gain some of the benefits of expert design through online learning.

We have conducted some preliminary experiments in this area. A CTW-based method was used to predict the high-level actions chosen online by ρUCT. This learnt distribution replaced our previous uniformly random rollout policy. Figure 11 shows the results of using this learnt rollout policy on the cheese maze. The other domains we tested exhibited similar behaviour. Although more work remains, it is clear that even our current simple learning scheme can significantly improve the performance of ρUCT.

Although our first attempts have been promising, a more thorough investigation is required. It is likely that rollout policy learning methods for adversarial games, such as those investigated by Silver and Tesauro (2009), can be adapted to our setting. It would also be interesting to try to apply some form of search bootstrapping (Veness, Silver, Uther, & Blair, 2009) online. In addition, one could also look at ways to modify the UCB policy used in ρUCT to automatically take advantage of learnt rollout knowledge, similar to the heuristic techniques used in computer Go (Gelly & Silver, 2007).

9.2 Combining Mixture Environment Models

A key property of mixture environment models is that they can be *composed*. Given two mixture environment models ξ_1 and ξ_2, over model classes $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively, it is easy to show that the convex combination

$$\xi(x_{1:n} \,|\, a_{1:n}) := \alpha \xi_1(x_{1:n} \,|\, a_{1:n}) + (1-\alpha)\xi_2(x_{1:n} \,|\, a_{1:n})$$

is a mixture environment model over the union of $\mathcal{M}_1$ and $\mathcal{M}_2$. Thus there is a principled way for expanding the general predictive power of agents that use our kind of direct AIXI approximation.

9.3 Richer Notions of Context for FAC-CTW

Instead of using the most recent D bits of the current history h, the FAC-CTW algorithm can be generalised to use a set of D boolean functions on h to define the current context. We now formalise this notion, and give some examples of how this might help in agent applications.

Definition 12. *Let $\mathcal{P} = \{p_0, p_1, \ldots, p_m\}$ be a set of predicates (boolean functions) on histories $h \in (\mathcal{A} \times \mathcal{X})^n, n \geq 0$. A $\mathcal{P}$-model is a binary tree where each internal node is labeled with a predicate in $\mathcal{P}$ and the left and right outgoing edges at the node are labeled True and False respectively. A $\mathcal{P}$-tree is a pair $(M_\mathcal{P}, \Theta)$ where $M_\mathcal{P}$ is a $\mathcal{P}$-model and associated with each leaf node l in $M_\mathcal{P}$ is a probability distribution over $\{0, 1\}$ parametrised by $\theta_l \in \Theta$.*

A $\mathcal{P}$-tree $(M_\mathcal{P}, \Theta)$ represents a function g from histories to probability distributions on $\{0, 1\}$ in the usual way. For each history h, $g(h) = \theta_{l_h}$, where l_h is the leaf node reached by pushing h down the model $M_\mathcal{P}$ according to whether it satisfies the predicates at the internal nodes and $\theta_{l_h} \in \Theta$ is the distribution at l_h. The notion of a $\mathcal{P}$-context tree can now be specified, leading to a natural generalisation of Definition 8.

Both the Action-Conditional CTW and FAC-CTW algorithms can be generalised to work with $\mathcal{P}$-context trees in a natural way. Importantly, a result analogous to Lemma 2 can be established, which means that the desirable computational properties of CTW are retained. This provides a powerful way of extending the notion of context for agent applications. For example, with a suitable choice of predicate class $\mathcal{P}$, both prediction suffix trees (Definition 7) and looping suffix trees (Holmes & Jr, 2006) can be represented as $\mathcal{P}$-trees. It also opens up the possibility of using rich logical tree models (Blockeel & De Raedt, 1998; Kramer & Widmer, 2001; Lloyd, 2003; Ng, 2005; Lloyd & Ng, 2007) in place of prediction suffix trees.

9.4 Incorporating CTW Extensions

There are several noteworthy ways the original CTW algorithm can be extended. The finite depth limit on the context tree can be removed (Willems, 1998), without increasing the asymptotic space overhead of the algorithm. Although this increases the worst-case time complexity of generating a symbol from $O(D)$ to linear in the length of the history, the average-case performance may still be sufficient for good performance in the agent setting. Furthermore, three additional model classes, each significantly larger than the one used by CTW, are presented in the work of Willems, Shtarkov, and Tjalkens (1996). These could be made action conditional along the same lines as our FAC-CTW derivation. Unfortunately, online prediction with these more general classes is now exponential in the context depth D. Investigating whether these ideas can be applied in a more restricted sense would be an interesting direction for future research.

9.5 Parallelization of ρUCT

The performance of our agent is dependent on the amount of thinking time allowed at each time step. An important property of ρUCT is that it is naturally parallel. We have completed a prototype parallel implementation of ρUCT with promising scaling results using between 4 and 8 processing cores. We are confident that further improvements to our implementation will allow us to solve problems where our agent's planning ability is the main limitation.

9.6 Predicting at Multiple Levels of Abstraction

The FAC-CTW algorithm reduces the task of predicting a single percept to the prediction of its binary representation. Whilst this is reasonable for a first attempt at AIXI approximation, it's worth emphasising that subsequent attempts need not work exclusively at such a low level.

For example, recall that the FAC-CTW algorithm was obtained by chaining together l_X action-conditional binary predictors. It would be straightforward to apply a similar technique to chain together multiple k-bit action-conditional predictors, for $k > 1$. These k bits could be interpreted in many ways: e.g. integers, floating point numbers, ASCII characters or even pixels. This observation, along with the convenient property that mixture environment models can be composed, opens up the possibility of constructing more sophisticated, *hierarchical* mixture environment models.

10. Conclusion

This paper presents the first computationally feasible general reinforcement learning agent that *directly* and *scalably* approximates the AIXI ideal. Although well established theoretically, it has previously been unclear whether the AIXI theory could inspire the design of practical agent algorithms. Our work answers this question in the affirmative: empirically, our approximation achieves strong performance and theoretically, we can characterise the range of environments in which our agent is expected to perform well.

To develop our approximation, we introduced two new algorithms: ρUCT, a Monte-Carlo expectimax approximation technique that can be used with any online Bayesian approach to the general reinforcement learning problem and FAC-CTW, a generalisation of the powerful CTW algorithm to the agent setting. In addition, we highlighted a number of interesting research directions that could improve the performance of our current agent; in particular, model class expansion and the online learning of heuristic rollout policies for ρUCT.

We hope that this work generates further interest from the broader artificial intelligence community in both the AIXI theory and general reinforcement learning agents.

Acknowledgments

The authors thank Alan Blair, Thomas Degris-Dard, Evan Greensmith, Bernhard Hengst, Ramana Kumar, John Lloyd, Hassan Mahmud, Malcolm Ryan, Scott Sanner, Rich Sutton, Eric Wiewiora, Frans Willems and the anonymous reviewers for helpful comments and feedback. This work received support from the Australian Research Council under grant DP0988049. NICTA is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

References

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, *3*, 397–422.

Begleiter, R., El-Yaniv, R., & Yona, G. (2004). On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, *22*, 385–421.

Bertsekas, D. P., & Castanon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. *Journal of Heuristics*, *5*(1), 89–108.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, *101*(1-2), 285–297.

Boots, B., Siddiqi, S. M., & Gordon, G. J. (2010). Closing the learning-planning loop with predictive state representations. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pp. 1369–1370 Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Brafman, R. I., & Tennenholtz, M. (2003). R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *AAAI*, pp. 1023–1028.

Chaslot, G.-B., Winands, M., Uiterwijk, J., van den Herik, H., & Bouzy, B. (2008a). Progressive strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, *4*(3), 343–357.

Chaslot, G. M., Winands, M. H., & Herik, H. J. (2008b). Parallel monte-carlo tree search. In *Proceedings of the 6th International Conference on Computers and Games*, pp. 60–71 Berlin, Heidelberg. Springer-Verlag.

Cover, T. M., & Thomas, J. A. (1991). *Elements of information theory*. Wiley-Interscience, New York, NY, USA.

Farias, V., Moallemi, C., Van Roy, B., & Weissman, T. (2010). Universal reinforcement learning. *Information Theory, IEEE Transactions on*, *56*(5), 2441 –2454.

Finnsson, H., & Björnsson, Y. (2008). Simulation-based approach to general game playing. In *AAAI*, pp. 259–264.

Gelly, S., & Silver, D. (2007). Combining online and offline learning in UCT. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 273–280.

Gelly, S., & Wang, Y. (2006). Exploration exploitation in Go: UCT for Monte-Carlo Go. In *NIPS Workshop on On-line trading of Exploration and Exploitation*.

Gelly, S., Wang, Y., Munos, R., & Teytaud, O. (2006). Modification of UCT with patterns in Monte-Carlo Go. Tech. rep. 6062, INRIA, France.

Hoehn, B., Southey, F., Holte, R. C., & Bulitko, V. (2005). Effective short-term opponent exploitation in simplified poker. In *AAAI*, pp. 783–788.

Holmes, M. P., & Jr, C. L. I. (2006). Looping suffix tree-based inference of partially observable hidden state. In *ICML*, pp. 409–416.

Hutter, M. (2002a). The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science.*, *13*(3), 431–443.

Hutter, M. (2002b). Self-optimizing and Pareto-optimal policies in general environments based on Bayes-mixtures. In *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence. Springer.

Hutter, M. (2005). *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer.

Hutter, M. (2007). Universal algorithmic intelligence: A mathematical top→down approach. In *Artificial General Intelligence*, pp. 227–290. Springer, Berlin.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1995). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *ECML*, pp. 282–293.

Kramer, S., & Widmer, G. (2001). Inducing classification and regression trees in first order logic. In Džeroski, S., & Lavrač, N. (Eds.), *Relational Data Mining*, chap. 6. Springer.

Krichevsky, R., & Trofimov, V. (1981). The performance of universal coding. *IEEE Transactions on Information Theory*, *IT-27*, 199–207.

Kuhn, H. W. (1950). A simplified two-person poker. In *Contributions to the Theory of Games*, pp. 97–103.

Legg, S., & Hutter, M. (2004). Ergodic MDPs admit self-optimising policies. Tech. rep. IDSIA-21-04, Dalle Molle Institute for Artificial Intelligence (IDSIA).

Legg, S. (2008). *Machine Super Intelligence*. Ph.D. thesis, Department of Informatics, University of Lugano.

Levin, L. A. (1973). Universal sequential search problems. *Problems of Information Transmission*, *9*, 265–266.

Li, M., & Vitányi, P. (2008). *An Introduction to Kolmogorov Complexity and Its Applications* (Third edition). Springer.

Littman, M., Sutton, R., & Singh, S. (2002). Predictive representations of state. In *NIPS*, pp. 1555–1561.

Lloyd, J. W. (2003). *Logic for Learning: Learning Comprehensible Theories from Structured Data*. Springer.

Lloyd, J. W., & Ng, K. S. (2007). Learning modal theories. In *Proceedings of the 16th International Conference on Inductive Logic Programming*, LNAI 4455, pp. 320–334.

Makino, T. (2009). Proto-predictive representation of states with simple recurrent temporal-difference networks. In *ICML*, pp. 697–704.

McCallum, A. K. (1996). *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, University of Rochester.

Ng, K. S. (2005). *Learning Comprehensible Theories from Structured Data*. Ph.D. thesis, The Australian National University.

Pankov, S. (2008). A computational approximation to the AIXI model. In *AGI*, pp. 256–267.

Poland, J., & Hutter, M. (2005). Defensive universal learning with experts. In *Proc. 16th International Conf. on Algorithmic Learning Theory*, Vol. LNAI 3734, pp. 356–370. Springer.

Poland, J., & Hutter, M. (2006). Universal learning of repeated matrix games. Tech. rep. 18-05, IDSIA.

Poupart, P., & Vlassis, N. (2008). Model-based bayesian reinforcement learning in partially observable domains. In *ISAIM*.

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete bayesian reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pp. 697–704 New York, NY, USA. ACM.

Rissanen, J. (1983). A universal data compression system. *IEEE Transactions on Information Theory*, *29*(5), 656–663.

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, *25*(2), 117–150.

Rosencrantz, M., Gordon, G., & Thrun, S. (2004). Learning low dimensional predictive representations. In *Proceedings of the twenty-first International Conference on Machine Learning*, p. 88 New York, NY, USA. ACM.

Ross, S., Chaib-draa, B., & Pineau, J. (2008). Bayes-adaptive POMDPs. In Platt, J., Koller, D., Singer, Y., & Roweis, S. (Eds.), *Advances in Neural Information Processing Systems 20*, pp. 1225–1232. MIT Press, Cambridge, MA.

Schmidhuber, J., Zhao, J., & Wiering, M. A. (1997). Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, *28*, 105–130.

Schmidhuber, J. (1997). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, *10*(5), 857–873.

Schmidhuber, J. (2002). The speed prior: A new simplicity measure yielding near-optimal computable predictions. In *Proc. 15th Annual Conf. on Computational Learning Theory*, pp. 216–228.

Schmidhuber, J. (2003). Bias-optimal incremental problem solving. In *Advances in Neural Information Processing Systems 15*, pp. 1571–1578. MIT Press.

Schmidhuber, J. (2004). Optimal ordered problem solver. *Machine Learning*, *54*, 211–254.

Shani, G. (2007). *Learning and Solving Partially Observable Markov Decision Processes*. Ph.D. thesis, Ben-Gurion University of the Negev.

Shani, G., & Brafman, R. (2004). Resolving perceptual aliasing in the presence of noisy sensors. In *NIPS*.

Silver, D., & Tesauro, G. (2009). Monte-carlo simulation balancing. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 945–952 New York, NY, USA. ACM.

Silver, D., & Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*. To appear.

Singh, S., James, M., & Rudary, M. (2004). Predictive state representations: A new theory for modeling dynamical systems. In *UAI*, pp. 512–519.

Solomonoff, R. J. (1964). A formal theory of inductive inference: Parts 1 and 2. *Information and Control*, *7*, 1–22 and 224–254.

Strehl, A. L., Li, L., & Littman, M. L. (2009). Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research*, *10*, 2413–2444.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., & Littman, M. L. (2006). PAC model-free reinforcement learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pp. 881–888 New York, NY, USA. ACM.

Strens, M. (2000). A Bayesian framework for reinforcement learning. In *ICML*, pp. 943–950.

Suematsu, N., & Hayashi, A. (1999). A reinforcement learning algorithm in partially observable environments using short-term memory. In *NIPS*, pp. 1059–1065.

Suematsu, N., Hayashi, A., & Li, S. (1997). A Bayesian approach to model learning in non-Markovian environment. In *ICML*, pp. 349–357.

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

Sutton, R. S., & Tanner, B. (2004). Temporal-difference networks. In *NIPS*.

Tjalkens, T. J., Shtarkov, Y. M., & Willems, F. M. J. (1993). Context tree weighting: Multi-alphabet sources. In *Proceedings of the 14th Symposium on Information Theory Benelux*.

Veness, J., Ng, K. S., Hutter, M., & Silver, D. (2010). Reinforcement Learning via AIXI Approximation. In *Proceedings of the Conference for the Association for the Advancement of Artificial Intelligence (AAAI)*.

Veness, J., Silver, D., Uther, W., & Blair, A. (2009). Bootstrapping from Game Tree Search. In *Neural Information Processing Systems (NIPS)*.

Wang, T., Lizotte, D. J., Bowling, M. H., & Schuurmans, D. (2005). Bayesian sparse sampling for on-line reward optimization. In *ICML*, pp. 956–963.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.

Willems, F., Shtarkov, Y., & Tjalkens, T. (1997). Reflections on "The Context Tree Weighting Method: Basic properties". *Newsletter of the IEEE Information Theory Society*, *47*(1).

Willems, F. M. J. (1998). The context-tree weighting method: Extensions. *IEEE Transactions on Information Theory*, *44*, 792–798.

Willems, F. M. J., Shtarkov, Y. M., & Tjalkens, T. J. (1996). Context weighting for general finite-context sources. *IEEE Trans. Inform. Theory*, *42*, 42–1514.

Willems, F. M., Shtarkov, Y. M., & Tjalkens, T. J. (1995). The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory*, *41*, 653–664.