Reinforcement Learning via AIXI Approximation

Joel Veness
University of NSW & NICTA
JOELV@CSE.UNSW.EDU.AU

Kee Siong Ng
Medicare Australia & ANU
KEESIONG.NG@GMAIL.COM

Marcus Hutter
ANU & NICTA
MARCUS.HUTTER@ANU.EDU.AU

David Silver
University College London
DAVIDSTARSILVER@GOOGLEMAIL.COM

Abstract

This paper introduces a principled approach for the design of a scalable general reinforcement learning agent. This 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 Monte Carlo Tree Search algorithm along with an agent-specific extension of the Context Tree Weighting algorithm. Empirically, we present a set of encouraging results on a number of stochastic, unknown, and partially observable domains.

1 Introduction

Consider an agent that exists within some unknown environment. The agent interacts with the environment in cycles. At each cycle, the agent executes an action and receives in turn an observation and a reward. The *general reinforcement learning problem* is to construct an agent that, over time, collects as much reward as possible from an initially *unknown* environment.

The AIXI agent (Hutter 2005) is a formal, mathematical solution to the general reinforcement learning problem. It can be decomposed into two main components: planning and prediction. Planning amounts to performing an expectimax operation to determine each action. Prediction uses Bayesian model averaging, over the largest possible model class expressible on a Turing Machine, to predict future observations and rewards based on past experience. AIXI is shown in (Hutter 2005) to be optimal in the sense that it will rapidly learn an accurate model of the unknown environment and exploit it to maximise its expected future reward.

As AIXI 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. This paper demonstrates, for the first time, how a practical agent can be built from the AIXI theory. Our solution directly approximates the planning and prediction components of AIXI. In particular, we use a generalisation of UCT (Kocsis and Szepesvári 2006) to approximate the expectimax operation, and an agent-specific extension of CTW (Willems, Shtarkov, and Tjalkens 1995), a Bayesian model averaging algorithm for prediction suffix trees, for prediction and learning. Perhaps surprisingly, this kind of direct approximation is possible, practical and theoretically appealing. Importantly, the essential characteristic of AIXI, its generality, can be largely preserved.

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 environment.

The (finite) action, observation, and reward spaces are denoted by $\mathcal{A}, \mathcal{O}$, and $\mathcal{R}$ respectively. An observation-reward pair *or* is called a percept. We use $\mathcal{X}$ to denote the percept space $\mathcal{O} \times \mathcal{R}$.

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

Notation: A string $x_1 x_2 \ldots x_n$ of length n is denoted by $x_{1:n}$. The empty string is denoted by ϵ. The concatenation of two strings s and r is denoted by sr. 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 $a_1 x_1 a_2 x_2 \ldots a_n x_n$ and $ax_{<j}$ denotes $a_1 x_1 a_2 x_2 \ldots a_{j-1} x_{j-1}$.

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

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

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

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

Equation 1, called the chronological condition in (Hutter 2005), captures the natural constraint that action a_n has no effect on observations made before it. For convenience, we drop the index n in ρ_n from here onwards.

Given an environment ρ,

$$\rho(x_n \,|\, a x_{<n} a_n) := \frac{\rho(x_{1:n} \,|\, a_{1:n})}{\rho(x_{<n} \,|\, a_{<n})} \quad (2)$$

is the ρ-probability of observing x_n in cycle n given history $h = ax_{<n}a_n$, provided $\rho(x_{<n} \,|\, a_{<n}) > 0$. It now follows that

$$\rho(x_{1:n} \,|\, a_{1:n}) = \rho(x_1 \,|\, a_1)\rho(x_2 \,|\, a x_1 a_2) \cdots \rho(x_n \,|\, a x_{<n} a_n). \quad (3)$$

Definition 2 is used to describe both the true (but unknown) underlying environment and the agent's *subjective* model of the environment. The latter is called the agent's *environment model* and is typically learnt from data. Definition 2 is extremely general. It captures a wide variety of environments, including standard reinforcement learning setups such as MDPs and POMDPs.

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 $m \in \mathbb{N}$. Formally, a policy is a function that maps a history to an action. The expected future value of an agent acting under a particular policy is defined as follows.

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*

$$v_\rho^m(\pi, ax_{1:t}) := \mathbb{E}_{x_{t+1:t+m}\sim\rho}\left[\sum_{i=t+1}^{t+m} R_i(ax_{\leq t+m})\right], \tag{4}$$

where for $t+1 \leq k \leq t+m$, $a_k := \pi(ax_{<k})$, and $R_k(aor_{\leq t+m}) := r_k$. 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 $h \equiv ax_{1:t} \in (\mathcal{A}\times\mathcal{X})^t$, then

$$V_\rho^m(h) = \max_{a_{t+1}} \sum_{x_{t+1}} \rho(x_{t+1}\,|\,ha_{t+1}) \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \rho(x_{t+m}\,|\,hax_{t+1:t+m-1}a_{t+m}) \left[\sum_{i=t+1}^{t+m} r_i\right]. \tag{5}$$

We will refer to Equation 5 as the *expectimax operation*. The m-horizon optimal action a_{t+1}^* at time $t+1$ is related to the expectimax operation by

$$a_{t+1}^* = \arg\max_{a_{t+1}} V_\rho^m(ax_{1:t}a_{t+1}) \tag{6}$$

Eqs 4 and 5 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

In the general reinforcement learning setting, the environment ρ is unknown to the agent. One way to learn an environment model is to take a Bayesian approach. Instead of committing to any single 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 above procedure is similar to Bayesian methods for predicting sequences of (singly typed) observations. The key difference in the agent setup is that each prediction is now also dependent on previous agent actions. We incorporate this by using the *action-conditional* definitions and identities of Section 2.

Definition 4. *Given a model class $\mathcal{M} := \{\rho_1, \rho_2, \ldots\}$ 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 $\xi(x_{1:n}\,|\,a_{1:n}) := \sum_{\rho\in\mathcal{M}} w_0^\rho \rho(x_{1:n}\,|\,a_{1:n})$.*

The next result follows immediately.

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

Proposition 1 allows us to use a mixture environment model whenever we can use an environment model. Its importance will become clear shortly.

To make predictions using a mixture environment model ξ, we use

$$\xi(x_n\,|\,ax_{<n}a_n) = \frac{\xi(x_{1:n}\,|\,a_{1:n})}{\xi(x_{<n}\,|\,a_{<n})}, \tag{7}$$

which follows from Proposition 1 and Eq. 2. The RHS of Eq. 7 can be written out as a convex combination of model predictions to give

$$\xi(x_n\,|\,ax_{<n}a_n) = \sum_{\rho\in\mathcal{M}} w_{n-1}^\rho \rho(x_n\,|\,ax_{<n}a_n), \tag{8}$$

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

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

Bayesian agents enjoy a number of strong theoretical performance guarantees; these are explored in Section 6. In practice, the main difficulty in using a mixture environment model is computational. A rich model class is required if the mixture environment model is to possess general prediction capabilities, however naively using (8) for online prediction requires at least $O(|\mathcal{M}|)$ time to process each new piece of experience. One of our main contributions, introduced in Section 5, is a large, efficiently computable mixture environment model that runs in time $O(\log(\log|\mathcal{M}|))$. Before looking at that, we will examine in the next section a Monte Carlo Tree Search algorithm for approximating the expectimax operation.

4 Monte Carlo Expectimax Approximation

Full-width computation of the expectimax operation (5) takes $O(|\mathcal{A}\times\mathcal{X}|^m)$ time, which is unacceptable for all but tiny values of m. This section introduces ρUCT, a generalisation of the popular UCT algorithm (Kocsis and Szepesvári 2006) that can be used to approximate a finite horizon expectimax operation given an environment model ρ. The key idea of Monte Carlo search is to sample observations from the environment, rather than exhaustively considering all possible observations. This allows for effective planning in environments with large observation spaces. Note that since an environment model subsumes both MDPs and POMDPs, ρUCT effectively extends the UCT algorithm to a wider class of problem domains.

The UCT algorithm has proven effective in solving large discounted or finite horizon MDPs. It assumes a generative model of the MDP 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 the 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 $\rho(or \mid h)$ to predict the next percept. The main subtlety with this extension is that the history used to determine the conditional probabilities must be updated *during the search* to reflect the extra information an agent will have at a hypothetical future point in time.

We will use Ψ to represent all the nodes in the search tree, $\Psi(h)$ to represent the node corresponding to a particular history h, $\hat{V}_\rho^m(h)$ to represent the sample-based estimate of the expected future reward, and $T(h)$ to denote the number of times a node $\Psi(h)$ has been sampled. Nodes corresponding to histories that end or do not end with an action are called chance and decision nodes respectively.

Algorithm 1 describes the top-level algorithm, which the agent calls at the beginning of each cycle. It is initialised with the agent's total experience h (up to time t) and the planning horizon m. It repeatedly invokes the SAMPLE routine until out of time. Importantly, ρUCT is an *anytime* algorithm; an approximate best action, whose quality improves with time, is always available. This is retrieved by BESTACTION, which computes $a_t^* = \arg\max_{a_t} \hat{V}_\rho^m(ax_{<t}a_t)$.

Algorithm 1 ρUCT(h, m)

Require: A history h
Require: A search horizon $m \in \mathbb{N}$

```
1: INITIALISE(Ψ)
2: repeat
3:     SAMPLE(Ψ, h, m)
4: until out of time
5: return BESTACTION(Ψ, h)
```

Algorithm 2 describes the recursive routine used to sample a single future trajectory. It uses the SELECTACTION routine to choose moves at interior nodes, and invokes the ROLLOUT routine at unexplored leaf nodes. The ROLLOUT routine picks actions uniformly at random 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. Notice that the recursive calls on Lines 6 and 11 append the most recent percept or action to the history argument.

Algorithm 3 describes the UCB (Auer 2002) policy used to select actions at decision nodes. The α and β constants denote the smallest and largest elements of $\mathcal{R}$ respectively. The parameter C varies the selectivity of the search; larger values grow 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 found.

Algorithm 2 SAMPLE(Ψ, h, m)

Require: A search tree Ψ
Require: A history h
Require: A remaining search horizon $m \in \mathbb{N}$

```
1: if m = 0 then
2:     return 0
3: else if Ψ(h) is a chance node then
4:     Generate (o, r) from ρ(or | h)
5:     Create node Ψ(hor) if T(hor) = 0
6:     reward ← r + SAMPLE(Ψ, hor, m − 1)
7: else if T(h) = 0 then
8:     reward ← ROLLOUT(h, m)
9: else
10:    a ← SELECTACTION(Ψ, h, m)
11:    reward ← SAMPLE(Ψ, ha, m)
12: end if
13: V̂(h) ← 1/(T(h)+1) [reward + T(h)V̂(h)]
14: T(h) ← T(h) + 1
15: return reward
```

The ramifications of the ρUCT extension are particularly significant to Bayesian agents described in Section 3. Proposition 1 allows ρUCT to be instantiated with a mixture environment model, which directly incorporates model uncertainty 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 Bayes-optimal 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.

Algorithm 3 SELECTACTION(Ψ, h, m)

Require: A search tree Ψ
Require: A history h
Require: A remaining search horizon $m \in \mathbb{N}$
Require: An exploration/exploitation constant $C > 0$

```
1: U = {a ∈ A : T(ha) = 0}
2: if U ≠ {} then
3:     Pick a ∈ U uniformly at random
4:     Create node Ψ(ha)
5:     return a
6: else
7:     return arg max_{a∈A} { 1/(m(β−α)) V̂(ha) + C sqrt( log(T(h)) / T(ha) ) }
8: end if
```

5 Action-Conditional CTW

We now introduce a large mixture environment model for use with ρUCT. Context Tree Weighting (CTW) (Willems, Shtarkov, and Tjalkens 1995) is an efficient and theoretically well-studied binary sequence prediction algorithm that works well in practice. It is an online Bayesian model averaging algorithm that computes, at each time point t, the

probability

$$\Pr(y_{1:t}) = \sum_M \Pr(M) \Pr(y_{1:t} \mid M), \tag{10}$$

where $y_{1:t}$ is the binary sequence seen so far, M is a prediction suffix tree (Ron, Singer, and Tishby 1996), $\Pr(M)$ is the prior probability of M, and the summation is over *all* prediction suffix trees of bounded depth D. A naive computation of (10) takes time $O(2^{2^D})$; using CTW, this computation requires only $O(D)$ time. In this section, we outline how CTW can be extended to compute probabilities of the form

$$\Pr(x_{1:t} \mid a_{1:t}) = \sum_M \Pr(M) \Pr(x_{1:t} \mid M, a_{1:t}), \tag{11}$$

where $x_{1:t}$ is a percept sequence, $a_{1:t}$ is an action sequence, and M is a prediction suffix tree as in (10). This extension allows CTW to be used as a mixture environment model (Definition 4) in the ρUCT algorithm, where we combine (11) and (2) to predict the next percept given a history.

Krichevsky-Trofimov Estimator. We start with a brief review of the KT estimator for Bernoulli distributions. Given a binary string $y_{1:t}$ with a zeroes and b ones, the KT estimate of the probability of the next symbol is given by

$$\Pr_{kt}(Y_{t+1} = 1 \mid y_{1:t}) := \frac{b + 1/2}{a + b + 1}. \tag{12}$$

The KT estimator can be 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. The probability of a string $y_{1:t}$ is given by

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

Prediction Suffix Trees. We next describe prediction suffix trees. We consider a binary tree where all the left edges are labelled 1 and all the right edges are labelled 0. The depth of a binary tree M is denoted by $d(M)$. Each node in M can be identified by a string in $\{0,1\}^*$ as usual: ϵ represents the root node of M; and if $n \in \{0,1\}^*$ is a node in M, then $n1$ and $n0$ represent respectively the left and right children of node n. The set of M's leaf nodes is denoted by $L(M) \subset \{0,1\}^*$. Given a binary string $y_{1:t}$ where $t \geq d(M)$, we define $M(y_{1:t}) := y_t y_{t-1} \dots y_{t'}$, where $t' \leq t$ is the (unique) positive integer such that $y_t y_{t-1} \dots y_{t'} \in L(M)$.

Definition 5. *A prediction suffix tree (PST) is a pair (M, Θ), where M is a binary tree and associated with each $l \in L(M)$ is a distribution over $\{0,1\}$ parameterised by $\theta_l \in \Theta$. We call M the model of the PST and Θ the parameter of the PST.*

A PST (M, Θ) maps each binary string $y_{1:t}$, $t \geq d(M)$, to $\theta_{M(y_{1:t})}$; 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 in Figure 1 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.

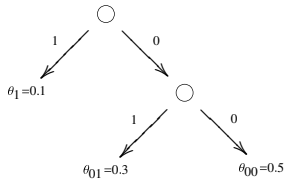

Figure 1: An example prediction suffix tree

Action-Conditional PST. 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. Further, we only ever condition on actions; this is achieved by appending bit representations of actions to the input sequence without updating the PST parameters.

Assume $|\mathcal{X}| = 2^{l_\mathcal{X}}$ for some $l_\mathcal{X} > 0$. Denote by $[\![x]\!] = x[1, l_\mathcal{X}] = x[1]x[2]\dots x[l_\mathcal{X}]$ the bit representation of $x \in \mathcal{X}$. Denote by $[\![x_{1:t}]\!] = [\![x_1]\!][\![x_2]\!]\dots[\![x_t]\!]$ the bit representation of a sequence $x_{1:t}$. Action symbols are treated similarly.

To do action-conditional sequence prediction using a PST with a given model M but unknown parameter, we start with $\theta_l := \Pr_{kt}(1 \mid \epsilon) = 1/2$ at each $l \in L(M)$. We set aside an initial portion of the binary history sequence 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 distribution $\theta_{M(h)}$;
   (b) observe the next bit $x[i]$, update $\theta_{M(h)}$ using (12) according to the value of $x[i]$, and then set $h := hx[i]$.

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

$$h_{M,n} := h_{i_1} h_{i_2} \cdots h_{i_k} \tag{13}$$

where $1 \leq i_1 < i_2 < \cdots < i_k \leq t$ and, for each i, $i \in \{i_1, i_2, \dots i_k\}$ iff h_i is a percept bit and n is a prefix of $M(h_{1:i-1})$. We have the following expression for the probability of $x_{1:t}$ given M and $a_{1:t}$:

$$\begin{aligned}\Pr(x_{1:t} \mid M, a_{1:t}) &= \prod_{i=1}^{t} \prod_{j=1}^{l_\mathcal{X}} \Pr(x_i[j] \mid M, [\![a x_{<i} a_i]\!] x_i[1, j-1]) \\ &= \prod_{n \in L(M)} \Pr_{kt}(h_{M,n}).\end{aligned} \tag{14}$$

Context Tree Weighting. The above deals with action-conditional prediction using a single PST. We now show how we can efficiently perform action-conditional prediction using a Bayesian mixture of PSTs. There are two main computational tricks: the use of a data structure to represent all PSTs of a certain maximum depth and the use of probabilities of sequences in place of conditional probabilities.

Definition 6. *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 weighted probability P_w^n of each node n in the context tree T after seeing $h := [\![ax_{1:t}]\!]$ is defined as follows:

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

The following is a straightforward extension of a result due to (Willems, Shtarkov, and Tjalkens 1995).

Lemma 1. *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*

$$P_w^n = \sum_{M \in C_{D-d}} 2^{-\Gamma_{D-d}(M)} \prod_{l \in L(M)} \Pr_{kt}(h_{T,nl}), \tag{15}$$

where C_d is the set of all models of PSTs with depth $\leq d$, and $\Gamma_d(M)$ is the code-length for M given by the number of nodes in M minus the number of leaf nodes in M of depth d.

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

$$P_w^\epsilon(x_{1:t}|a_{1:t}) = \sum_{M \in C_D} 2^{-\Gamma_D(M)} \prod_{l \in L(M)} \Pr_{kt}(h_{T,l}) \tag{16}$$

$$= \sum_{M \in C_D} 2^{-\Gamma_D(M)} \prod_{l \in L(M)} \Pr_{kt}(h_{M,l}) \tag{17}$$

$$= \sum_{M \in C_D} 2^{-\Gamma_D(M)} \Pr(x_{1:t} \mid M, a_{1:t}), \tag{18}$$

where the last step follows from (14). Notice that the prior $2^{-\Gamma_D(\cdot)}$ penalises PSTs with large tree structures. The conditional probability of x_t given $ax_{<t}a_t$ can be obtained from (2). We can also efficiently sample the individual bits of x_t one by one.

Computational Complexity. The Action-Conditional CTW algorithm grows the context tree dynamically. Using a context tree with depth D, there are at most $O(tD\log(|\mathcal{O}||\mathcal{R}|))$ nodes in the context tree after t cycles. In practice, this is a lot less than 2^D, the number of nodes in a fully grown context tree. The time complexity of Action-Conditional CTW is also impressive, requiring $O(D\log(|\mathcal{O}||\mathcal{R}|))$ time to process each new piece of agent experience and $O(mD\log(|\mathcal{O}||\mathcal{R}|))$ to sample a single trajectory when combined with ρUCT. Importantly, this is independent of t, which means that the computational overhead does not increase as the agent gathers more experience.

6 Theoretical Results

Putting the ρUCT and Action-Conditional CTW algorithms together yields our approximate AIXI agent. We now investigate some of its properties.

Model Class Approximation. By instantiating (5) with the mixture environment model (18), one can show that the optimal action for an agent at time t, having experienced $ax_{<t}$, is given by

$$\arg\max_{a_t} \sum_{x_t} \cdots \max_{a_{t+m}} \sum_{x_{t+m}} \left[\sum_{i=t}^{t+m} r_i\right] \sum_{M \in C_D} 2^{-\Gamma_D(\rho)} \Pr(x_{1:t+m} \mid M, a_{1:t+m}).$$

Compare this to the action chosen by the AIXI agent

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

where class $\mathcal{M}$ consists of all computable environments ρ and $K(\rho)$ denotes the Kolmogorov complexity of ρ. Both use a prior that favours simplicity. The main difference is in the subexpression describing the mixture over the model class. AIXI uses a mixture over all enumerable chronological semimeasures, which is completely general but incomputable. Our approximation uses a mixture of all prediction suffix trees of a certain maximum depth, which is still a rather general class, but one that is efficiently computable.

Consistency of ρUCT. (Kocsis and Szepesvári 2006) shows that the UCT algorithm is consistent in finite horizon MDPs and derive finite sample bounds on the estimation error due to sampling. By interpreting histories as Markov states, the general reinforcement learning problem reduces to a finite horizon MDP and the results of (Kocsis and Szepesvári 2006) are now directly applicable. Restating the main consistency result in our notation, we have

$$\forall \epsilon \forall h \lim_{T(h)\to\infty} \Pr\left(|V_\rho^m(h) - \hat{V}_\rho^m(h)| \leq \epsilon\right) = 1. \tag{19}$$

Furthermore, the probability that a suboptimal action (with respect to $V_\rho^m(\cdot)$) is chosen by ρUCT goes to zero in the limit.

Convergence to True Environment. The next result, adapted from (Hutter 2005), shows that if there is a good model of the (unknown) environment in C_D, then Action-Conditional CTW will predict well.

Theorem 1. *Let μ be the true environment, and $\Upsilon \equiv P_w^\epsilon$ the mixture environment model formed from (18). The μ-expected squared difference of μ and Υ is bounded as follows. For all $n \in \mathbb{N}$, for all $a_{1:n}$,*

$$\sum_{k=1}^{n} \sum_{x_{<k}} \mu(x_{<k} \mid a_{<k}) \sum_{x_k} \Big(\mu(x_k \mid ax_{<k}a_k) - \Upsilon(x_k \mid ax_{<k}a_k)\Big)^2$$

$$\leq \min_{M \in C_D} \Big\{ \Gamma_D(M)\ln 2 + KL(\mu(\cdot \mid a_{1:n}) \,||\, \Pr(\cdot \mid M, a_{1:n})) \Big\}, \tag{20}$$

where $KL(\cdot \,||\, \cdot)$ is the KL divergence of two distributions.

If the RHS of (20) is finite over all n, then the sum on the LHS can only be finite if Υ converges sufficiently fast to μ. If KL grows sublinear in n, then Υ still converges to μ (in a weaker Cesaro sense), which is for instance the case for all k-order Markov and all stationary processes μ.

Overall Result. Theorem 1 above in conjunction with (Hutter 2005, Thm.5.36) imply $V_\Upsilon^m(h)$ converges to $V_\mu^m(h)$ as long as there exists a model in the model class that approximates the unknown environment μ well. This, and the consistency (19) of the ρUCT algorithm, imply that $\hat{V}_\Upsilon^m(h)$ converges to $V_\mu^m(h)$. More detail can be found in (Veness et al. 2009).

7 Experimental Results

This section evaluates our approximate AIXI agent on a variety of test domains. The Cheese Maze, 4x4 Grid and Extended Tiger domains are taken from the POMDP literature. The TicTacToe domain comprises a repeated series of games against an opponent who moves randomly. The Biased RockPaperScissor domain is described in (Farias et al. 2007), which involves the agent repeatedly playing RockPaperScissor against an exploitable opponent. Two more challenging domains are included: Kuhn Poker (Hoehn et al. 2005), where the agent plays second against a Nash optimal player and a partially observable version of Pacman described in (Veness et al. 2009). With the exception of Pacman, each domain has a known optimal solution. Although our domains are modest, requiring the agent to learn the environment from scratch *significantly* increases the difficulty of each of these problems.

Domain	$\|\mathcal{A}\|$	$\|\mathcal{O}\|$	$\mathcal{A}$ / $\mathcal{O}$ / $\mathcal{R}$ bits	D	m
Cheese Maze	4	16	2 / 4 / 5	96	8
Tiger	3	3	2 / 2 / 7	96	5
4×4 Grid	4	1	2 / 1 / 1	96	12
TicTacToe	9	19683	4 / 18 / 3	64	9
Biased RPS	3	3	2 / 2 / 2	32	4
Kuhn Poker	2	6	1 / 4 / 3	42	2
Pacman	4	65536	2 / 16 / 8	64	8

Table 1: Parameter Configuration

Table 1 outlines the parameters used in each experiment. The sizes of the action and observation spaces are given, along with the number of bits used to encode each space. The context depth parameter D specifies the maximal number of recent bits used by the Action-Conditional CTW prediction scheme. The search horizon is given 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 an appropriate compromise between these two competing dimensions for each problem domain.

Figure 2 shows how the performance of the agent scales with experience, measured in terms of number of interaction cycles. Experience was gathered by a decaying ϵ-greedy policy, which chose randomly or used ρUCT. The results are normalised with respect to the optimal average reward per time step, except in Pacman, where we normalised to an estimate. Each data point was obtained by starting the agent with an amount of experience given by the x-axis and running it greedily for 2000 cycles. The amount of search used for each problem domain, measured by the number of ρUCT simulations per cycle, is given in Table 2. (The average search time per cycle is also given.) The agent converges to optimality on all the test domains with known optimal values, and exhibits good scaling properties on our challenging Pacman variant. Visual inspection[1] of Pacman shows that the agent, whilst not playing perfectly, has already learnt a number of important concepts.

Table 2 summarises the resources required for approximately optimal performance on our test domains. Timing

[1] http://www.youtube.com/watch?v=RhQTWidQQ8U

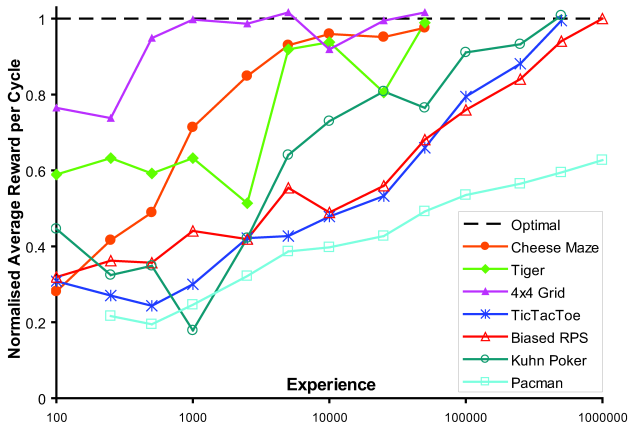

Figure 2: Learning scalability results

Domain	Experience	Simulations	Search Time
Cheese Maze	5×10^4	500	0.9s
Tiger	5×10^4	10000	10.8s
4×4 Grid	2.5×10^4	1000	0.7s
TicTacToe	5×10^5	5000	8.4s
Biased RPS	1×10^6	10000	4.8s
Kuhn Poker	5×10^6	3000	1.5s

Table 2: Resources required for optimal performance

statistics were collected on an Intel dual 2.53Ghz Xeon. Domains that included a planning component such as Tiger required more search. Convergence was somewhat slower in TicTacToe; the main difficulty for the agent was learning not to lose the game immediately by playing an illegal move. Most impressive was that the agent learnt to play an approximate best response strategy for Kuhn Poker, without knowing the rules of the game or the opponent's strategy.

8 Related Work

The BLHT algorithm (Suematsu and Hayashi 1999) is closely related to our work. It 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.

The Active-LZ (Farias et al. 2007) algorithm 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. We implemented the Active-LZ test domain, Biased RPS, and compared against their published results. Our agent was able to achieve optimal levels of performance within 10^6 cycles; in contrast, Active-LZ was still suboptimal after 10^8 cycles.

U-Tree (McCallum 1996) is an online agent algorithm that attempts to discover a compact state representation from a raw stream of experience. 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 learned state representation, which is then used by the agent for action selection.

It is instructive to compare and contrast our AIXI approximation with the Active-LZ and U-Tree algorithms. 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 has the potential to 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 Action-Conditional 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 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 applicability to the general reinforcement learning problem.

Our work is also related to Bayesian Reinforcement Learning. In model-based Bayesian RL (Poupart and Vlassis 2008; Strens 2000), a distribution over (PO)MDP parameters is maintained. In contrast, we maintain an exact Bayesian mixture of PSTs. The ρUCT algorithm shares similarities with Bayesian Sparse Sampling (Wang et al. 2005); the key differences are estimating the leaf node values with a rollout function and guiding the search with the UCB policy.

A more comprehensive discussion of related work can be found in (Veness et al. 2009).

9 Limitations

The main limitation of our current AIXI approximation is the restricted model class. 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 identification of *efficient* and *general* model classes that better approximate the AIXI ideal is an important area for future work. Some preliminary ideas are explored in (Veness et al. 2009).

10 Conclusion

We have introduced the first computationally tractable approximation to the AIXI agent and shown that it provides a promising approach to the general reinforcement learning problem. Investigating multi-alphabet CTW for prediction, parallelisation of ρUCT, further expansion of the model class (ideally, beyond variable-order Markov models) or more sophisticated rollout policies for ρUCT are exciting areas for future investigation.

11 Acknowledgements

This work received support from the Australian Research Council under grant DP0988049. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Auer, P. 2002. Using confidence bounds for exploitation-exploration trade-offs. *JMLR* 3:397–422.

Farias, V.; Moallemi, C.; Weissman, T.; and Roy, B. V. 2007. Universal Reinforcement Learning. *CoRR* abs/0707.3087.

Hoehn, B.; Southey, F.; Holte, R. C.; and Bulitko, V. 2005. Effective short-term opponent exploitation in simplified poker. In *AAAI'05*, 783–788.

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

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *ECML*, 282–293.

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

Poupart, P., and Vlassis, N. 2008. Model-based Bayesian Reinforcement Learning in Partially Observable Domains. In *ISAIM*.

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

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

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

Veness, J.; Ng, K. S.; Hutter, M.; and Silver, D. 2009. A Monte Carlo AIXI Approximation. *CoRR* abs/0909.0801.

Wang, T.; Lizotte, D.; Bowling, M.; and Schuurmans, D. 2005. Bayesian sparse sampling for on-line reward optimization. In *ICML*, 956–963.

Willems, F. M.; Shtarkov, Y. M.; and Tjalkens, T. J. 1995. The Context Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory* 41:653–664.