Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Context Tree Maximizing Reinforcement Learning

Phuong Nguyen[1,2] and **Peter Sunehag**[1] and **Marcus Hutter**[1,2,3]
[1]Australian National University and [2]NICTA and [3]ETHZ
nmphuong@cecs.anu.edu.au, first.last@anu.edu.au

Abstract

Recent developments in reinforcement learning for non-Markovian problems witness a surge in history-based methods, among which we are particularly interested in two frameworks, ΦMDP and MC-AIXI-CTW. ΦMDP attempts to reduce the general RL problem, where the environment's states and dynamics are both unknown, to an MDP, while MC-AIXI-CTW incrementally learns a mixture of context trees as its environment model. The main idea of ΦMDP is to connect generic reinforcement learning with classical reinforcement learning. The first implementation of ΦMDP relies on a stochastic search procedure for finding a tree that minimizes a certain cost function. This does not guarantee finding the minimizing tree, or even a good one, given limited search time. As a consequence it appears that the approach has difficulties with large domains. MC-AIXI-CTW is attractive in that it can incrementally and analytically compute the internal model through interactions with the environment. Unfortunately, it is computationally demanding due to requiring heavy planning simulations at every single time step. We devise a novel approach called CTMRL, which analytically and efficiently finds the cost-minimizing tree. Instead of the context-tree weighting method that MC-AIXI-CTW is based on, we use the closely related context-tree maximizing algorithm that selects just one single tree. This approach falls under the ΦMDP framework, which allows the replacement of the costly planning component of MC-AIXI-CTW with simple Q-Learning. Our empirical investigation shows that CTMRL finds policies of quality as good as MC-AIXI-CTW's on six domains including a challenging Pacman domain, but in an order of magnitude less time.

1 Introduction

Reinforcement Learning (RL) is an area of active research in artificial intelligence in which agents learn a task through interactions with the environment. Markov Decision Processes (MDP) is the most well-studied framework in RL. Nevertheless, in the real world, most environments are non-Markovian due to partial observability, making them particularly challenging to deal with. Recently Hutter (2009) introduced a framework for generic reinforcement learning called ΦMDP, where the central idea is to find the best map Φ that transforms a general RL (GRL) problem to an MDP. The term "best" was formalized in a cost criterion that represents a trade-off between the resulting model's size and its ability to predict the outcome of following a given policy. Being able to predict the consequences of actions is key to being able to choose them well. The first empirical study of ΦMDP conducted by Nguyen, Sunehag, and Hutter (2011) showed encouraging results. However, their use of stochastic search does not guarantee finding a good tree given limited search time and it does not scale well to large domains. In another direction, MC-AIXI-CTW (Veness et al. 2010) models the environment as a Bayesian mixture of context trees and uses UCT Monte Carlo planning (Kocsis and Szepesvári 2006) to choose its actions. MC-AIXI-CTW learns its mixture model incrementally as it interacts with the environment. However, the major disadvantage with MC-AIXI-CTW is that it has to do hundreds or thousands of planning simulations at every single time step in order to find the best action. We develop a novel algorithm called CTMRL that combines ideas from ΦMDP and MC-AIXI-CTW. CTMRL analytically and efficiently computes its state set as the agent interacts with the environment. It is, therefore, able to work with large domains that exhibit partial observability and have millions of underlying states.

Related work. Beside ΦMDP and MC-AIXI-CTW, other context-tree based methods in the RL literature include Active-LZ (Farias et al. 2010), U-tree (McCallum 1996), and BLHT (Suematsu, Hayashi, and Li 1997; Suematsu and Hayashi 1999). Active-LZ extends the Lempel-Ziv compression scheme to RL, while U-tree employs a local criterion derived from the Kolmogorov-Smirnov test for building a context tree. BLHT like MC-AIXI-CTW, models the environment as a Bayesian mixture of tree sources, but unlike MX-AIXI-CTW it performs a MAP estimate to find one tree. This makes it similar to our algorithm. They used dynamic programming to choose its actions based on the estimated model. For this and other reasons, BLHT needs further development in the direction of what is presented in this article to scale to larger domains. Apart from the context tree-based methods, POMDP learning (Chrisman 1992) and predictive state representation (Littman, Sutton, and Singh 2002; McCracken and Bowling 2005; Boots and Gordon 2010) are alternative approaches. In particular the latter is currently an active research direction in the RL community where there has been recent progress in making the

approach practical, though it is still a challenge to scale to larger domains. Our aim in this paper is to further develop the history-based (context tree) approaches since these have shown promise in large domains like Car Driving (McCallumn 1994) and Pacman (Veness et al. 2011).

Context Tree Maximizing for Reinforcement Learning (CTMRL). Context Tree maximizing (CTM) (Willems, Shtarkov, and Tjalkens 2000) is an attractive approach to sequence prediction that analytically calculates the optimal context-tree model in the sense of the Minimum Description Length principle (MDL) (Rissanen 1978). We extend this methodology for tackling the GRL problem. We combine the CTM method with Value Iteration and Q-learning to create our own algorithm called CTMRL. The main steps of CTMRL can be briefly described as follows: A binarized history is defined as the one obtained from binarizing the original history of observations, actions and rewards. Given the binarized history, we apply the CTM procedure to find the smallest context-tree model that can predict the next percept (observation and reward) well. To enhance the learning quality, the process is repeated after obtaining more experience from the environment.

Contributions. Our contributions are: (1) extending the CTM approach to RL; (2) presenting a theorem that shows the connection between the CTMRL procedure and the ΦMDP cost function; and (3) demonstrating the practicality of CTMRL through empirical investigation.

Paper organization. Section 2 contains background material. Section 3 describes our extension of CTM to RL. Section 4 presents an empirical investigation on six domains. Finally, Section 5 contains the conclusions.

2 Background

2.1 Markov Decision Process (MDP)

An MDP (Sutton and Barto 1998) is a decision process in which at any discrete time t, given action a_t, the probability of the next observation and reward o_{t+1} and r_{t+1}, given the past history $h_t = a_1 o_2 r_2 \dots a_{t-1} o_t r_t$, only depends on the current observation o_t. That is, $P(o_{t+1}, r_{t+1}|h_t a_t) = P(o_{t+1}, r_{t+1}|o_t, a_t)$. Observations in this process are called states of the environment. Formally, a finite MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ in which $\mathcal{S}$ is a finite state set; $\mathcal{A}$ is a finite action set; $T = (T^a_{ss'} : s, s' \in \mathcal{S},\ a \in \mathcal{A})$ is a collection of transition probabilities of the next state $s_{t+1} = s'$ given the current state $s_t = s$ and action $a_t = a$; and $R = (R^a_{ss'} : s, s' \in \mathcal{S},\ a \in \mathcal{A})$ is a reward function $R^a_{ss'} = \mathbf{E}[r_{t+1}|s_t = s, a_t = a, s_{t+1} = s']$. The return at time step t is the total discounted reward $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$, where γ is a geometric discount factor ($0 \leq \gamma < 1$).

The action value in state s following policy π is defined as $Q^\pi(s,a) = \mathbf{E}_\pi[R_t|s_t = s, a_t = a] = \mathbf{E}_\pi[\sum_{k=0}^\infty \gamma^k r_{t+k+1}|s_t = s, a_t = a]$. For a known MDP, Action-Value Iteration (AVI) is a useful algorithm to find an estimate of the optimal action values $Q^* := \max_\pi Q^\pi$. The approach is based on the optimal action-value Bellman equation (Sutton and Barto 1998), and iterates the update $Q(s,a) \leftarrow \sum_{s'} T^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q(s',a')]$. For unknown MDPs, an efficient solver is Q-learning, which incrementally adjusts action values through the update $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha_t(s_t,a_t)err_t$ where $err_t = r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)$ is the instant feedback error, and $\alpha_t(s_t,a_t)$ is the learning rate at time t.

2.2 Context Trees

Definition. A context tree or suffix tree is a tree data structure that represents suffixes of a given string of symbols (Weiner 1973; Nguyen, Sunehag, and Hutter 2011). In the RL context, the string is the history. We define several versions of context trees that will be employed in the subsequent development of our method. If the set of symbols are the observation set, we call the context tree an Observation Context Tree (OCT). If the sets of symbols for odd depths and even depths of the tree are observation and action sets respectively, the tree is named Action-Observation Context Tree (AOCT). If the set of symbols is the instance set $\mathcal{I} = \{aor : a \in \mathcal{A}, o \in \mathcal{O}, r \in \mathcal{R}\}$ where $\mathcal{A}, \mathcal{O}, \mathcal{R}$ are action, observation and reward sets respectively, we call the tree Instance Context Tree (ICT).

The state suffix set, or briefly state set $\mathcal{S} = \{s^1, s^2, \dots, s^m\}$ induced from a context tree $\mathcal{T}$ is defined as the set of all possible strings of edge labels (symbols) forming along a path from a leaf node to the root node of $\mathcal{T}$. When we refer to a state or context $s \in \mathcal{S}$, it also equivalently means the leaf node corresponding to $s \in \mathcal{T}$. Tree $\mathcal{T}$ and its suffix set $\mathcal{S}$ are considered to be equivalent as well since they can be constructed from each other. The two terms contexts and states are also used interchangeably.

2.3 Context Tree Maximizing (CTM) for Binary Sequence Prediction

One of the fundamental problems of sequence prediction is to find the smallest tree source from which future observations can be well-predicted. In this section, we summarize the main results from Willems, Shtarkov, and Tjalkens (2000) who describe the CTM algorithm for binary sequence prediction. The minimum description length (with two-part codes) (Rissanen 1978) or minimum message length (Wallace and Boulton 1968) of a sequence $x_{1:n} \in \{0,1\}^n$ over the models $\mathcal{S} \in \mathcal{C}_D$ is defined as

$$\Lambda_D(x_{1:n}) = \min_{\mathcal{S} \in \mathcal{C}_D} \left[\log \frac{1}{P_c(x_{1:n}|\mathcal{S})} + \Gamma_D(\mathcal{S}) \right] \quad (1)$$

given any sequence $x_{1-D:0}$ of (fictitious) past symbols, which we have suppressed in the expression. Here $\mathcal{C}_D$ is the set of all binary context trees with depth not greater than D. Note that there are over 2^{2^D} trees in $\mathcal{C}_D$, so brute-force evaluation of (1) is impossible for interesting D; the CTM algorithm can determine (1) in time $O(D \times n)$. $\Gamma_D(\mathcal{S}) := |\mathcal{S}| - 1 + |\{s : s \in \mathcal{S},\ l(s) \neq D\}|$ is a code length of tree $\mathcal{S}$, here $|\mathcal{S}|$ is the cardinality of the suffix set $\mathcal{S}$ and $l(s)$ is the length of s; $x_{1:n}$ refers to $x_1 \dots x_n$; and P_c stands for coding probability of $x_{1:n}$ with respect to the suffix tree $\mathcal{S}$. It is defined as $P_c(x_{1:n}|S) = \prod_{s \in \mathcal{S}} P_e(a_s, b_s)$, where $a_s = a_s(x_{1:n})$ and $b_s = b_s(x_{1:n})$ are respectively the number of zeros and the number of ones that occur in

$x_{1:n}$ at instants τ for which $x_{\tau-l(s):\tau-1} = s$; and P_e denotes the estimate probability. The coding scheme utilized for the description is arithmetic coding (Cover and Thomas 1991). $P_e(a,b)$ is the KT-estimator (Krichevsky and Trofimov 1981), the block probability estimate of seeing a zeros and b ones. The KT estimate of the predictive probability of the next bit given a binary string $x_{1:t}$ with a zeros and b ones is $P_{kt}(X_{t+1} = 1|x_{1:t}) := \frac{b+1/2}{a+b+1}$, which implies

$$P_e(a,b) = \frac{\frac{1}{2}(1+\frac{1}{2})\dots(a-\frac{1}{2})\frac{1}{2}(1+\frac{1}{2})\dots(b-\frac{1}{2})}{(a+b)!}.$$

P_e can be incrementally computed as $P_e(a+1,b) = P_e(a,b)\times(a+1/2)/(a+b+1)$. The KT estimator is based on the Bayesian-probability framework by assuming a Jeffrey prior $P(\theta) \propto \theta^{-1/2}(1-\theta)^{-1/2}$ $(Beta(1/2,1/2))$ on the parameter $\theta \in [0,1]$ of the Bernoulli distribution.

For each node s, we assign a maximizing "probability"

$$P^D_{m,s} := \begin{cases} \frac{1}{2}\max(P_e(a_s,b_s), P^D_{m,0s}P^D_{m,1s}) & \text{if } l(s) < D \\ P_e(a_s,b_s) & \text{if } l(s) = D \end{cases}$$

and maximizing set

$$\mathcal{S}^D_{m,s} := \begin{cases} \mathcal{S}^D_{m,0s}\times 0 \cup \mathcal{S}^D_{m,1s}\times 1 & \text{if } P_e(a_s,b_s) < \\ & P^D_{m,0s}P^D_{m,1s} \\ & \text{and } l(s) < D, \\ \{\epsilon\} & \text{otherwise} \end{cases}$$

where ϵ is the empty string.

The intuition behind the CTM recursive formulas can be briefly explained as follows: If $l(s) = D$, $P_e(a_s,b_s)$ should serve as a good coding probability as we assume that the tree source is memoryless at depth D. If an internal node s of the tree is memoryless, then again $P_e(a_s,b_s)$ is a good coding probability. Otherwise, if s is not memoryless, we have to further recursively split state s until the memoryless contexts are reached to find a good coding probability. As the maximizing probability is defined, the good coding probability for a non-memoryless context s should be the product of $P^D_{m,0s}$ and $P^D_{m,1s}$. The multiplicative factor $\frac{1}{2}$ in the maximizing-probability definition is to represent the uniform model prior. All models $\mathcal{S} \in \mathcal{C}_D$ are assumed to have the prior $P(\mathcal{S}) = 2^{-\Gamma_D(\mathcal{S})}$, which represents the philosophy of Occam's razor (Li and Vitani 2008) - small models are preferable.

An important property CTM has in common with CTW is that the computational complexity is linear in the sequence length. The CTM procedure is described as follows: At time t, the agent sees observation o_t, and it has to traverse down from the CTM tree's root to the leaf node corresponding to the context $s^D_t = o_{t-D:t-1}$. In each step, either we create a new node with parameters initialized if this node's context is not represented in the current CTM tree yet; or we incrementally update the counts and estimate probability $(a_s,\ b_s,\ P_e(a_s,b_s))$ of the existing node corresponding to the current context in the CTM tree. Then we travel in reverse from the leaf node $(s^D_t = o_{t-D:t-1})$ to the root node $(s^0_t = \epsilon)$ in order to update the maximizing probabilities $(P^D_{m,s})$ for all nodes or contexts $s^d_t = o_{t-d:t-1},\ d = D,\dots,0$. When updating each $P^D_{m,s}$ with $l(s) < D$, we need to know both $P_{m,0s}$ and $P_{m,1s}$. If either of the nodes 0s or 1s does not exist in the current CTM tree (0s and 1s are unseen contexts), its maximizing probability is equal to the prior probability $\frac{1}{2}$.

The maximizing probability and the maximizing set corresponding to a node $s \in \mathcal{S}$ are denoted by $P^D_{m,s}(x_{1:n})$ and $\mathcal{S}^D_{m,s}(x_{1:n})$ respectively. For any sequence $x_{1-D:0}$ of past symbols, define the maximizing coding distribution as $P_m(x_{1:n}) := P^D_{m,\epsilon}(x_{1:n})$, and the maximizing state set as $\mathcal{S}_m(x_{1:n}) := \mathcal{S}^D_{m,\epsilon}(x_{1:n})$. We have the following theorem, which shows the connection of the CTM procedure to (1). Namely, that the minimizer in (1) is found through this procedure.

Theorem 1 *(Willems, Shtarkov, and Tjalkens 2000) For any sequence $x_{1:n} \in \{0,1\}^n$, we have*

$$\begin{aligned} & \log\frac{1}{P_m(x_{1:n})} \\ = \ & \log\frac{1}{P_c(x_{1:n}|\mathcal{S}_m(x_{1:n}))} + \Gamma_D(\mathcal{S}_m(x_{1:n})) \\ = \ & \Lambda_D(x_{1:n}) \end{aligned}$$

given any sequence $x_{1-D:0}$ of past symbols.

3 Context Tree Maximizing for Reinforcement Learning

The primary goal of an RL agent is to find the optimal policy π for taking action a_t given the history $h_t = a_0o_1r_1a_1\dots o_{t-1}r_{t-1}a_{t-1}o_tr_t$ so as to maximize the total long-term reward. We focus on attacking the challenging GRL problem where the agent has no prior knowledge on the environment's model including what the states are.

Our primary purpose is to develop an analytic and efficient procedure for finding a good MDP state set given a history. Thereby, the GRL problem is reduced to solving the found MDP for which a rich literature of classical RL methods is available (Sutton and Barto 1998; Csaba 2010). It is worth recalling that this reduction falls under the ΦMDP framework (Hutter 2009). We will base our algorithm on the original ΦMDP cost function **Cost**$(\Phi|h)$ and present recursive formulas that analytically find the minimizer of the cost criterion. A key difference to the sequence prediction setting is that, as in Hutter (2009), we condition on the actions since we want to be able to predict the consequences of different choices for them. The cost is a trade-off between this predictive ability and the size of the model and can be viewed as an extension of the MDL principle to RL. The main obstacle when using the resulting algorithm for large problems is finding a reliable estimate of the multivariate coding probability. To overcome this issue, we borrow a binary factorization technique from Veness et al. (2011).

Cost function. We define the following cost function for a model $\mathcal{S}$ in the set of all ICTs with depths not greater than

the given maximal depth D (denoted as $\mathcal{C}_D$) to be

$$\begin{aligned}&\textbf{Cost}(\mathcal{S}|h_n,h_0)\\ =\;&\log\frac{1}{P_c(h_n|a_{0:n-1},h_0,\mathcal{S})}+\Gamma_D(\mathcal{S})\\ =\;&\log\frac{1}{P_c(s_{1:n},r_{1:n}|a_{0:n-1},h_0,\mathcal{S})}+\Gamma_D(\mathcal{S})\quad(2)\end{aligned}$$

where $h_0 = a_{-D}o_{1-D}r_{1-D}\ldots\ a_{-1}o_0r_0$ is any (fictitious) initial history; $\Gamma_D(\mathcal{S}) := |\mathcal{S}|-1+|\{s : s\in\mathcal{S}, l(s)\neq D\}|$ is the model penalty of $\mathcal{S}$ with respect to model class $\mathcal{C}_D$; and $s_t = \Phi_{\mathcal{S}}(h_t),\ t = 1,\ldots,n$ with $\Phi_{\mathcal{S}}$ being the map extracting suffix of the history h_t that matches a suffix or a state in the state set $\mathcal{S}$; and $h_0 = a_{-D}o_{1-D}r_{1-D}\ldots\ a_{-1}o_0r_0$ is the given initial history. We aim to minimize (2) to find the model with the shortest description length, which is the optimal model that we seek. The interested reader is referred to Nguyen, Sunehag, and Hutter (2011) for full details of the context-tree mapping from histories to states of an MDP.

Denote the instance set of $\mathcal{C}_D$ as $\mathcal{I} := \{x^1,\ldots,x^{|\mathcal{I}|}\} = \{aor : a\in\mathcal{A}, o\in\mathcal{O}, r\in\mathcal{R}\}$; at time t, $x_t = a_{t-1}o_tr_t$. Note that $P_c(s_{1:n},r_{1:n}|a_{0:n-1},h_0,\mathcal{S}) = P_c(o_{1:n},r_{1:n}|a_{0:n-1},h_0,\mathcal{S}) = \prod_a\prod_s P_e^{x|sa}$ where $P_e^{x|sa} := P_e^{x|sa}(n_{x^1},n_{x^2},\ldots,n_{x^{|\mathcal{I}|}})$ is the block probability estimate of seeing a sequence containing n_{x^i} of x^i, $i = 1,\ldots,|\mathcal{I}|$, given that action a is taken at context s. Hence,

$$\textbf{Cost}(\mathcal{S}|h_n,h_0) = \Sigma_a\Sigma_s\log\frac{1}{P_e^{x|sa}}+\Gamma_D(\mathcal{S})$$

Recursive formulas for minimizing cost. We now describe the analytic formulas that compute the minimizer of (2). The maximizing probability at state s is recursively defined as follows:

$$P_{m,s}^D := \begin{cases}\frac{1}{2}\max\left(\prod_{a\in\mathcal{A}}P_e^{x|sa},\prod_i P_{m,x^is}^D\right) & \text{if } l(s)<D\\ \prod_{a\in\mathcal{A}}P_e^{x|sa} & \text{if } l(s)=D\end{cases}$$

The maximizing state set $S_{m,s}^D$ is defined as

$$\mathcal{S}_{m,s}^D := \begin{cases}\bigcup_{x^i}\mathcal{S}_{m,x^is}^D\times x^i & \text{if } \prod_{a\in\mathcal{A}}P_e^{x|sa} < \prod_{a\in\mathcal{A}}\prod_i P_{m,x^is}^D \\ & \text{and } l(s)<D,\\ \{\epsilon\} & \text{otherwise}\end{cases}$$

The intuitive idea behind the above maximizing probability and maximizing state set is similar to the binary CTM. The main difference here is that the formulas are defined for the non-binary case and with the condition on actions in order to represent the coding probability given that the environment model is an MDP with state set $\mathcal{S}$. The maximizing state set at each state is defined by conditioning on actions. This is the reason why we have the condition on actions in the definition of P_e for each state. Given a memoryless state s, and an action a taken at s, $P_e^{x|sa}$ should serve as a good coding probability for s. Hence, the good coding probability for a memoryless context s should be the product of $P_e^{x|sa}$ over all possible actions. The details are represented in the recursive formula of the maximizing probability.

It can be shown that $S_{m,s}^D = \{s' : s's\in\mathcal{S}_{m,\epsilon}^D\}$ and we shall see from the theorem below that $\mathcal{S}_{m,\epsilon}^D$ is indeed the minimizer of (2). By induction and with similar proof techniques as presented in Willems, Shtarkov, and Tjalkens (2000), we can prove the following lemma.

Lemma 2 *For any state s, we have*

$$\begin{aligned}P_{m,s}^D &= 2^{-\Gamma_{D-d}(\mathcal{S}_{m,s}^D)}\prod_{a\in\mathcal{A}}\prod_{u\in\mathcal{S}_{m,s}^D}P_e^{x|usa}\\ &= \max_{\mathcal{U}\in\mathcal{C}_{D-d}}2^{-\Gamma_{D-d}(\mathcal{U})}\prod_{a\in\mathcal{A}}\prod_{u\in\mathcal{U}}P_e^{x|usa}\end{aligned}$$

where $l(s) = d$.

From the lemma, we can see that the maximizing probability of a state s is the product of the coding probability of the minimal memoryless tree rooted at node s, and the model prior of the minimal tree at context s. If s is memoryless, the minimal tree is the empty tree. If $s = \epsilon$, the minimal tree is the optimal solution of (2) that we seek. Now define $P_m(h_n|a_{0:n-1},h_0) := P_{m,\epsilon}^D(h_n|a_{0:n-1},h_0)$ and $S_m(h_n|a_{0:n-1},h_0) := S_{m,\epsilon}^D(h_n|a_{0:n-1},h_0)$; and $\Lambda_D(h_n|a_{0:n-1},h_0)$ as the minimum of (2). The following theorem can be straightforwardly shown by setting $s = \epsilon$ in Lemma 2.

Theorem 3 *Given a history h_n up to time n, and an initial history h_0, we have*

$$\begin{aligned}&\log\frac{1}{P_m(h_n|a_{0:n-1},h_0)}\\ =\;&\log\frac{1}{P_c(h_n|a_{0:n-1},h_0,\mathcal{S}_m(h_n|a_{0:n-1}h_0)))}+\\ &+\Gamma_D(\mathcal{S}_m(h_n|a_{0:n-1},h_0)))\\ =\;&\Lambda_D(h_n|a_{0:n-1},h_0))\end{aligned}$$

The theorem implies that the recursive procedure for computing maximizing probabilities and corresponding maximizing sets yield the optimal solution to the cost function (2). The computational complexity of this CTM procedure is $\mathcal{O}(D\times n)$, hence linear in the history length, while the model-space size is doubly exponential in D.

Binary Factorization. The main issue with what we have described so far is the estimation of $P_e^{x|sa}$. For large domains, it is difficult to get reliable estimates as we often do not have enough statistics based on type information. To alleviate the above drawback, we use a binarization and factorization approach inspired by Veness et al. (2011). First, actions, observations and rewards in the original history are binarized to obtain a binary history. Then for each bit of the percept (a percept is a pair of observation and reward), we use CTM to construct a separate binary tree to predict this bit. This strategy makes it possible to exploit the structure within each percept. That is, in domains with huge observation space, it is difficult to determine the most useful memory we need to remember in order to predict well at the type level as statistics are insufficient; however, at the bit level,

the much more abundant statistics allow us to learn the binary contexts to predict individual bits of the percept. From the union of learnt binary contexts, we then can determine the type context tree that has good-prediction capability.

The model class we consider here is the class of AOCTs. We believe that contexts including observations and actions are general enough to be widely applicable. The central idea of our approach is to find binary context trees for predicting each percept bit, then combine all the learnt contexts of these binary context trees to form an AOCT, which defines the MDP state set that we use. We next describe the technical details of the final resulting algorithm that we call CTMRL.

Suppose that l_a, l_o, and l_r are the minimum numbers of bits needed for binary representation of actions, observations and rewards respectively. For each symbol x, denote its binary representation as $[[x]] = x[1, l_x] = x[1]x[2]\dots x[l_x] \in \{0,1\}^{l_x}$. Denote the environment percept as $p := or$, and $[[p]] = [[or]] = [[o]][[r]] = p[1, l_p] = p[1]p[2]\dots p[l_p]$ where $l_p = l_o + l_r$. We consider joint models $M = (M_1, \dots, M_{l_p}) \in \mathcal{C}_D \times \dots \times \mathcal{C}_{D+l_p-1}$ where M_i is the action-conditional binary model for predicting bit i of the percept; and $\mathcal{C}_{D+i-1}$ is the set of all binary context trees with depth not greater than $D+i-1$ ($i = 1, \dots, l_p$). It should be noted here that except the first $i-1$ percept bits, contexts of each M_i comprise of binarized actions and observations only. Let $M^* := \text{argmin}_M$ **Cost**$(M|h_n, h_0)$ where the cost is defined as follows:

$$\begin{aligned}
& \textbf{Cost}(M|h_n, h_0) \\
= \;& \log \frac{1}{P_c(h_n|a_{0:n-1}, h_0, M)} + \sum_{i=1}^{p} \Gamma(M_i) \\
= \;& \sum_{t=1}^{n} \log \frac{1}{P_c(p_t|h_{t-1}a_{t-1}, h_0, M)} + \sum_{i=1}^{p} \Gamma(M_i) \\
= \;& \sum_{i=1}^{p} \left[\sum_{t=1}^{n} \log \frac{1}{P_c(p_t[i]|h_t^i, h_0, M_i)} + \Gamma(M_i) \right]
\end{aligned}$$

where $h_t^i = h_{t-1}a_{t-1}p_t[1 \dots i-1]$.

Hence the problem is reduced to finding the optimal M_i^*s ($i = 1, \dots, p$), which are obtained using the CTM procedure for the binary sequence prediction case as shown in Section 2.3. After $M^* = (M_1^*, \dots, M_p^*)$ is computed, the AOCT we ultimately use is $\widehat{S} = \bigcup_{i=1,\dots,l_p}^{\text{context}} M_i^*$, which is the smallest tree that contains all $S_{m,ap[1\dots i-1]}^{D+i-1}$ ($i = 1, \dots, l_p$ and for all a and p) as subtrees. Note that all $S_{m,ap[1\dots i-1]}^{D+i-1}$s are subtrees of M_i^* but rooted at context $ap[1 \dots i-1]$.

Algorithm. Based on the procedure for choosing a state set described above, we design an agent named CTMRL as specified in Algorithm 1. We first perform a certain number of random actions, then use this history to learn l_p CTMs that predict individual percept bits. Next, we unify the learnt binary contexts from the CTMs to form an AOCT. This AOCT is then employed to learn a good policy via AVI, and explore unseen scenarios through the Q-learning in the learning loop. The current history is then updated with the additional experiences gained from Q-learning. After that, we repeat the procedure but without the random actions. When the learning loop is finished, we may run Q-learning to further enhance the current policy. Note that as we are uncertain whether all possible observations and actions have been seen from the history, especially in large domains, an unseen symbol λ is added to $\mathcal{O}$ and $\mathcal{A}$ to represent unseen scenarios in the AOCT. In the Q-learning part of Algorithm 1, when an unseen context is encountered, this new state can be either added to the current AOCT, or lumped to the corresponding unseen context in the current AOCT. In all our experiments below, a lumping-state strategy is applied for the first five domains, while for Pacman, a state-adding feature is utilized as in such large domains, the number of unseen contexts can be significantly greater than the seen ones.

Algorithm 1 CTMRL

Require: $Environment$, $nLearningLoops$, n_is (n_i is the number of new experiences to collect in loop i), n_q (the iteration number in the ultimate Q-learning after the learning loop)

1: $i \leftarrow 0$
2: Create l_p empty CTMs, where the i^{th} CTM predicts the i^{th} bit of the percept p
3: $h \leftarrow$ initial random history obtained by performing n_0 random actions
4: $h' \leftarrow h$
5: **while** $i < nLearningLoops$ **do**
6: Update l_p CTMs based on history h'
7: Join learnt contexts from each of the CTMs to form AOCT $\mathcal{T}$
8: Compute frequency estimates $\widehat{M}$ of state transition and reward probabilities of the MDP model based on states induced from tree $\mathcal{T}$ and history h
9: Use AVI to find an estimate of optimal action values $\widehat{Q}$ based on $\widehat{M}$
10: (Optional) Evaluate the current optimal policy extracted from $\widehat{Q}$
11: $Q \leftarrow \widehat{Q} + \frac{R_{\max}}{1-\gamma}$ [[Optimistic Initialization]]
12: **if** $i < nLearningLoops - 1$ **then**
13: $h' \leftarrow$ Q-learning$(Q, \mathcal{S}^{\mathcal{T}}, \mathcal{A}, Environment, n_i)$
14: $h \leftarrow [h, h']$
15: **end if**
16: $i \leftarrow i + 1$
17: **end while**
18: $\widehat{Q}' \leftarrow$ Q-learning$(Q, \mathcal{S}^{\mathcal{T}}, \mathcal{A}, Environment, n_q)$
19: $\pi^*(s) \leftarrow \text{argmax}_a \widehat{Q}'(s, a)$ for all $s \in \mathcal{S}^{\mathcal{T}}$

Return π^*

The detailed working of the model learning steps in Algorithm 1 (lines 6 and 7) is best illustrated with an example. Given $\mathcal{O} = \{o^1, o^2\} = \{0, 1\}$, $\mathcal{A} = \{a^1, a^2\} = \{0, 1\}$ and $\mathcal{R} = \{0, 1, 2\}$, then $l_o = l_a = 1$, $l_r = 2$, and $l_p = l_o + l_r = 3$. Each observation or action is encoded as it is; while rewards in $\mathcal{R}$ are encoded as 00, 01,10 respectively. We analytically construct $l_p = 3$ CTMs to find the l_p optimal binary models M_1^*, M_2^*, and M_3^* to predict the three bits of the percept. Suppose that the three optimal

binary contexts found for predicting bits 1, 2, and 3 of the percept (conditioned on $ap[1 \dots i-1]$, with $i = 1, 2,$ and 3 respectively) are $\mathcal{S}_1^* = \{0, 01, 11\}$, $\mathcal{S}_2^* = \{00, 10, 1\}$, and $\mathcal{S}_3^* = \{0, 1\}$ ($\mathcal{S}_i^*$ is the minimal suffix set that has all $S^{D+i-1}_{m,ap[1\dots i-1]}$s, for all a and p, as its subtrees). From each of the learnt binary contexts (states) from $\mathcal{S}_1^*$, $\mathcal{S}_2^*$, and $\mathcal{S}_3^*$, we determine type contexts of the AOCT tree $\mathcal{T}$. $\mathcal{S}_1^*, \mathcal{S}_2^*$ and $\mathcal{S}_3^*$ respectively give $\{o^1, a^1o^2, a^2o^2\}$, $\{a^1o^1, a^2o^1, o^2\}$, $\{o^1, o^2\}$. Hence, $\mathcal{S}^{\mathcal{T}} = \{a^1o^1, a^2o^1, a^1o^2, a^2o^2\}$ is the minimal AOCT that has $\mathcal{S}_1^*$, $\mathcal{S}_2^*$ and $\mathcal{S}_3^*$ as its subtrees.

The purpose of the interplay between AVI and optimistic Q-learning is for efficient exploration of new contexts. In the learning loop, we initialize Q-learning with $\widehat{Q} + R_{\max}/(1-\gamma)$ where $\widehat{Q}$ is learnt by AVI. As a result, at the beginning the agent acts according to the policy extracted from $\widehat{Q}$. This will reinforce the structure of the current AOCT. Then as Q-learning updates values, the policy changes, and this allows the agent to explore unseen contexts. The learning loop is very helpful for the agent to gradually find the most useful MDP state set. In learning loop i we gather n_i new experiences from the environment in order to strengthen the current learnt contexts, and explore unseen scenarios. Small n_i allows careful learning of useful states at the expense of more computation mainly due to the AVI procedure. In small domains, with large n_is the agent may learn many states that are unimportant to determine the optimal policy.

Implementation for large domains (Pacman). In the Q-learning part of Algorithm 1, when an unseen context is encountered, this new state will be added to the current AOCT. The action value for the newly added state is initialized based on the value of the first subsequent seen state. Another issue to note here is that for large domains, a large number of experiences are needed to learn the environment model, and this will put high demand on memory allocation for the binary trees. Due to this issue, for each learning loop we delete the CTMs after the learnt binary contexts are added to the current AOCT; that is, all learnt binary contexts of CTMs in the current learning loop are retained in the AOCT, and CTMs must be constructed from scratch in the loop based on the history obtained from the previous loop. These modifications are vastly improving the memory efficiency of the algorithm, making it possible to run Pacman on a modest computer. There is, however, some loss of data efficiency.

We provide an example to illustrate the state-adding scenarios. Suppose the current AOCT is $\mathcal{S} = \{0, \lambda, 01, 11, \lambda 1\}$ where more than two observations can be received, and more than two actions are available. Then, if $o_t = 2$ is encountered, it will be added to $\mathcal{S}$; if the suffix 31 ($a_t = 3,\ o_{t+1} = 1$) is encountered, it will be added to $\mathcal{S}$ too; however, if the suffix 10 ($a_t = 1,\ o_{t+1} = 0$) or 20 ($a_t = 2,\ o_{t+1} = 0$) is encountered, they are not added to $\mathcal{S}$ since the existing state $s = 0$ can be extracted from either of the two suffixes. The action value for that new state is initialized based on the value of the first subsequent seen state. In our experiments this state-adding feature only applies to Pacman where the observation space is much larger than in the other five domains.

4 Experiments

This section presents our empirical investigation of Algorithm 1 for six domains, Cheese Maze, Tiger, Extended Tiger, Kuhn Poker, 4×4 Grid, and Pacman (see Veness et al. (2011) for the descriptions of these domains). The performance of our method, CTMRL, on the six domains is examined and compared with four competitors, ΦMDP, MC-AIXI-CTW, U-tree, and Active-LZ. In the last domain, Pacman, we only provide the results of CTMRL and MC-AIXI-CTW as U-tree and Active-LZ are unable to work with this large domain (Veness et al. 2011). ΦMDP as presented in Nguyen, Sunehag, and Hutter (2011) requires serious modification in the direction of CTMRL to deal with large observation spaces.

The parameters of Algorithm 1 are chosen as follows: For the first five small domains, $\gamma = 0.99$ (in Q-learning and AVI), $\eta = 0.1$ (in Q-learning), $D = 8 \times (l_a + l_o)$ (CTMs), and $(n_0, n_1, \dots, n_7) = (250, 250, 500, 1500, 2500, 5000, 15000, 25000)$. For Pacman, $\gamma = 0.9$ (in Q-learning and AVI), $\eta = 0.1$ (in Q-learning), $D = 2 \times (l_a + l_o)$ (CTMs) and $n_0 = 2500, n_1 = 2500, n_2 = 5000, n_i = 10000$ $(3 \le i \le 26)$, and $n_q = 97500000$. We use the results in Veness et al. (2011) and Nguyen, Sunehag, and Hutter (2011) to represent the methods that we compare CTMRL to. In each of the plots below, the learning quality of each algorithm is assessed at various time points by measuring average reward over 50000 actions. The exploration of each of the four competitors is temporarily switched off at such evaluation points. Performance of our algorithm is measured using the policy extracted from action values learned by AVI (step 10 of Algorithm 1). For Pacman, after time 250000 we run Q-learning on the fixed state set and we further evaluate the policy of our agent (the greedy policy with respect to the Q values) at some fixed points.

It can be seen from Figures 1, 3, 4, 5, 6, and 7; and Table 1 that CTMRL outperforms ΦMDP, U-tree, Active-LZ; and is competitive with MC-AIXI-CTW in terms of learning outcomes. However, compared to MC-AIXI-CTW, our approach has some clear advantages in computation time and memory. For example, in the cheese-maze domain, CTMRL consumes less than ten seconds to process 500 cycles of experience, and finds a near optimal policy; it also learnt a state set containing 22 states as represented in Figure 2. On Pacman, CTMRL, in less than a day, finds a policy, which is as good as MC-AIXI-CTW's given a week of computation on a stronger computer. Nevertheless, on this domain, MC-AIXI-CTW is much more data-efficient than CTMRL. The MC-AIXI-CTW experiments in Veness et al. (2011) were performed on a dual quad-core 2.53Ghz computer with 24GB of memory, while our machine is a dual core 2.4GHz with 2GB of memory.

5 Conclusions

We introduced the CTMRL algorithm for non-markovian RL based on the CTM sequence prediction algorithm. Overall, CTMRL is competitive with the state of the art MC-AIXI-CTW in terms of learning and superior to other com-

Domains	CTMRL			MC-AIXI-CTW			Optimal reward
	Experience	Total time	Max average reward	Experience	Search time	Max average reward	
Cheese maze	50000	27m	1.27	50000	12h30m	1.28	1.33
Tiger	50000	11m	1.07	50000	5h30h	1.12	1.084
Extended Tiger	50000	43m	4.58	50000	6h	3.97	4.99
Kuhn Poker	100000	14m	0.056	100000	1h25m	0.056	0.056
4×4 Grid	50000	29m	0.25	50000	7h30m	0.25	0.25
Pacman	100000000	18h	1.66	250000	168h	1.64	

Table 1: **Summary performance of CTMRL and MC-AIXI-CTW**

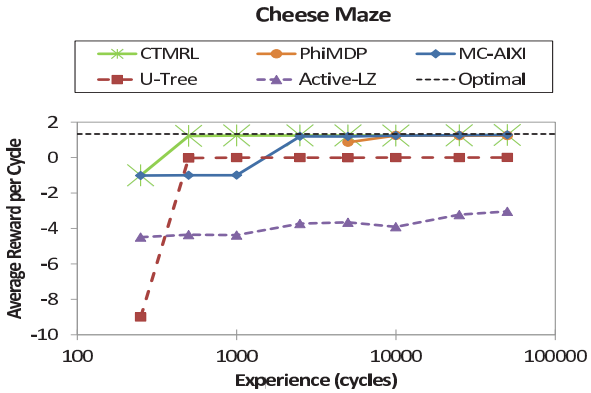

Figure 1: **Cheese maze**

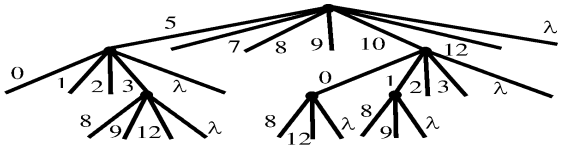

Figure 2: **Cheese maze tree**

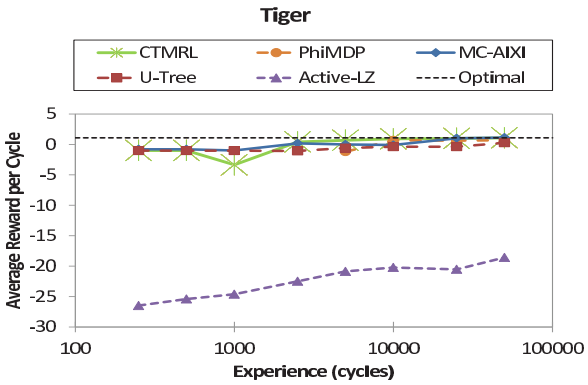

Figure 3: **Tiger**

petitors. Compared to MC-AIXI-CTW, CTMRL is dramatically more efficient in both computation time and memory. This is achieved by moving to the ΦMDP framework and thereby eliminating the need for costly online planning. For

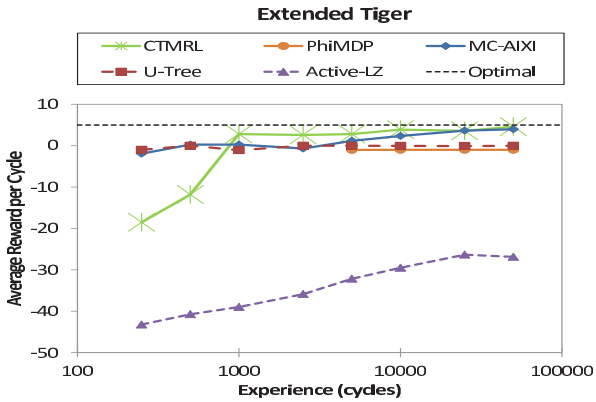

Figure 4: **Extended Tiger**

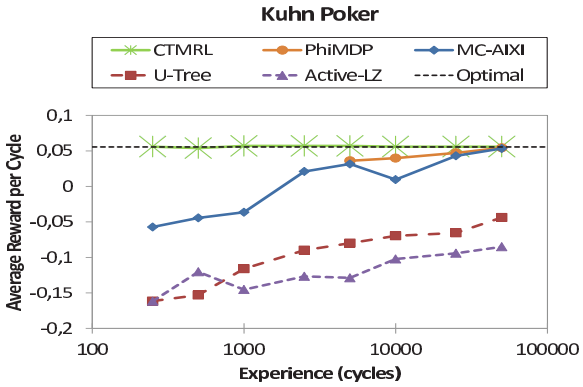

Figure 5: **Kuhn poker**

Pacman, we find a good policy in a day instead of a week, despite using a much weaker computer.

An intrinsic limitation of CTMRL and other context-tree based methods is that they are incapable of dealing with domains where long-term memory is crucial to take smart actions. That CTMRL and MC-AIXI-CTW can do well on the studied domains including Tiger and extended Tiger is mainly because context trees of a modest depth are able to capture all useful information.

Acknowledgement. This work was supported by ARC grant DP0988049 and by NICTA. We also thank Joel Veness and Daniel Visentin for their assistance with the experiments.

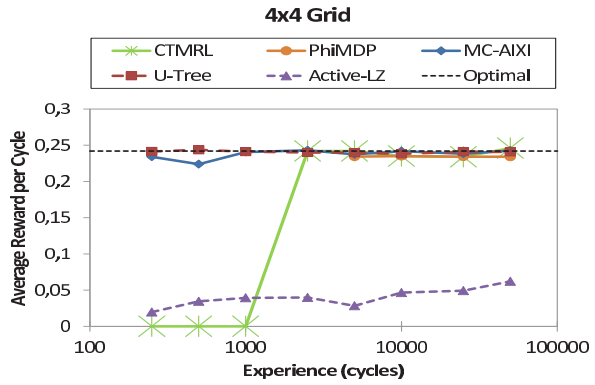

Figure 6: **4×4 grid**

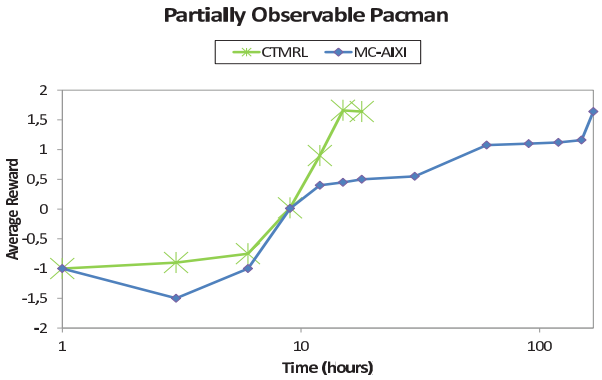

Figure 7: **CPU time on Pacman**

References

Boots, B., and Gordon, G. 2010. Predictive state temporal difference learning. In *The 24th Annual Conference on Neural Information Processing Systems*, 271–279.

Chrisman, L. 1992. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings Tenth National Conference on Artificial Intelligence*, 183–188. AAAI Press.

Cover, T. M., and Thomas, J. A. 1991. *Elements of Information Theory*. John Willey and Sons.

Csaba, S. 2010. Reinforcement learning algorithms for MDPs. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan and Claypool.

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

Hutter, M. 2009. Feature reinforcement learning: Part I. Unstructured MDPs. *Journal of General Artificial Intelligence* 1:3–24.

Kocsis, L., and Szepesvári. 2006. Bandit based Monte Carlo planing. In *European Conferernce in Machine Learning (ECML)*, 282–293.

Krichevsky, R., and Trofimov, V. 1981. The performance of universal coding. *IEEE Transactions on Information Theory* 27:199–207.

Li, M., and Vitani, P. 2008. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer.

Littman, M.; Sutton, R.; and Singh, S. 2002. Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, 1555–1561.

McCallum, R. A. 1996. *Reinforcemenet Learning with Selective Perception and Hidden State*. Ph.D. Dissertation, Department of Computer Science, University of Rochester.

McCallumn, R. A. 1994. Short-term memory in visual routines for "off-road car chasing". In *Working Notes of AAAI Spring Symposium Series, "Toward Physical Interaction and Manipulation"*.

McCracken, P., and Bowling, M. 2005. Online discovery and learning of predictive state representations. In *The 19th Annual Conference on Neural Information Processing Systems*, 875–882.

Nguyen, P.; Sunehag, P.; and Hutter, M. 2011. Feature reinforcement learning in practice. In *Proceedings of the 9th European Workshop in Reinforcement Learning*. Springer.

Rissanen, J. 1978. Modeling by shortest data description. *Automatica* 14:465–471.

Suematsu, N., and Hayashi, A. 1999. A reinforcement learning algorithm in partially observable environments using short-term memory. In *the 25th Annual Conference of Neural Information Processing Systems*, 349–357.

Suematsu, N.; Hayashi, A.; and Li, S. 1997. A bayesian approach to model learning in non-markovian environments. In *the 14th International Conference on Machine Learning*, 349–357.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning*. The MIT Press.

Veness, J.; Ng, K. S.; Hutter, M.; and Silver, D. 2010. Reinforcement learning via AIXI approximation. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 605–611. AAAI Press.

Veness, J.; Ng, K. S.; Hutter, M.; Uther, W.; and Silver, D. 2011. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research* 40(1):95–142.

Wallace, C. C., and Boulton, D. M. 1968. An information measure for classification. *1968* 11(2):185–194.

Weiner, P. 1973. Linear pattern matching algorithm. In *The 14th Annual IEEE Symposium on Switching and Automata Theory*, 1–11.

Willems, F. M. J.; Shtarkov, Y. M.; and Tjalkens, T. J. 2000. Context-tree maximizing. In *Conference on Information Sciences and Systems*. Princeton University.