Feature Reinforcement Learning in Practice

Phuong Nguyen, **Peter Sunehag** and **Marcus Hutter**

Research School of Computer Science, Australian National University
nmphuong@cecs.anu.edu.au
{peter.sunehag, marcus.hutter}@anu.edu.au

23 August 2011

Abstract

Following a recent surge in using history-based methods for resolving perceptual aliasing in reinforcement learning, we introduce an algorithm based on the feature reinforcement learning framework called ΦMDP [14]. To create a practical algorithm we devise a stochastic search procedure for a class of context trees based on parallel tempering and a specialized proposal distribution. We provide the first empirical evaluation for ΦMDP. Our proposed algorithm achieves superior performance to the classical U-tree algorithm [21] and the recent active-LZ algorithm [6], and is competitive with MC-AIXI-CTW [28] that maintains a bayesian mixture over all context trees up to a chosen depth. We are encouraged by our ability to compete with this sophisticated method using an algorithm that simply picks one single model, and uses Q-learning on the corresponding MDP. Our ΦMDP algorithm is much simpler, yet consumes less time and memory. These results show promise for our future work on attacking more complex and larger problems.

Contents

1 Introduction **3**

2 Preliminaries **4**

2.1 Markov Decision Processes (MDP) 4

2.2 Feature Reinforcement Learning 5

2.3 Context Trees 7

2.4 Stochastic search 9

3 The ΦMDP Algorithm **12**
3.1 Proposal Distribution for Stochastic Search over the Markov-AOCT Space 13

4 Experiments **15**
4.1 Experimental Setup 15
4.2 Environments and results 16

5 Conclusions **20**

Keywords

Reinforcement learning; MDP; MDL; feature selection; perceptual aliasing; coding.

1 Introduction

Reinforcement Learning (RL) [27] aims to learn how to succeed in a task through trial and error. This active research area is well developed for environments that are Markov Decision Processes (MDPs); however, real world environments are often partially observable and non-Markovian. The recently introduced Feature Markov Decision Process (ΦMDP) framework [14] attempts to reduce actual RL tasks to MDPs for the purpose of attacking the general RL problem where the environment's model as well as the set of states are unknown. In [26], Sunehag and Hutter take a step further in the theoretical investigation of Feature Reinforcement Learning by proving consistency results. In this article, we develop an actual Feature Reinforcement Learning algorithm and empirically analyze its performance in a number of environments.

One of the most useful classes of maps (Φs) that can be used to summarize histories as states of an MDP, is the class of context trees. Our stochastic search procedure, the principal component of our ΦMDP algorithm GSΦA, works on a subset of all context trees, called Markov trees. Markov trees have previously been studied in [22] but under names like FSMX sources or FSM closed tree sources. The stochastic search procedure employed for our empirical investigation utilizes a parallel tempering methodology [7], [12] together with a specialized proposal distribution. In the experimental section, the performance of the ΦMDP algorithm where stochastic search is conducted over the space of context-tree maps is shown and compared with three other related context tree-based methods.

Our ΦMDP algorithm is briefly summarized as follows. First, perform a certain number of random actions, then use this history to find a high-quality map by minimizing a cost function that evaluates the quality of each map. The quality here refers to the ability to predict rewards using the created states. We perform a search procedure for uncovering high-quality maps followed by executing Q-learning on the MDP whose states are induced by the detected optimal map. The current history is then updated with the additional experiences obtained from the interactions with the environment through Q-Learning. After that, we may repeat the procedure but without the random actions. The repetition refines the current "optimal" map, as longer histories provide more useful information for map evaluation. The ultimate optimal policy of the algorithm is retrieved from the action values Q on the resulting MDP induced from the final optimal map.

Contributions. Our contributions are: extending the original ΦMDP cost function presented in [14] to allow for more discriminative learning and more efficient minimization (through stochastic search) of the cost; identifying the Markov action-observation context trees as an important class of feature maps for ΦMDP; proposing the GSΦA algorithm where several chosen learning and search procedures are logically combined; providing the first empirical analysis of the ΦMDP model; and designing a specialized proposal distribution for stochastic search over the space of Markov trees, which is of critical importance for finding the best possible ΦMDP

agent.

Related Work. Our algorithm is a history-based method. This means that we are utilizing memory that in principle can be long, but in most of this article and in the related works is near term. Given a history h_t of observations, actions and rewards we define states $s_t = \Phi(h_t)$ based on some map Φ. The main class of maps that we will consider are based on context trees. The classical algorithm of this sort is U-tree [21], which uses a local criterion based on a statistical test for splitting nodes in a context tree; while ΦMDP employs a global cost function. Because of this advantage, ΦMDP can potentially be used in conjunction with any optimization methods to find the optimal model.

There has been a recent surge of interest in history based methods with the introduction of the active-LZ algorithm [6], which generalizes the widely used Lempel-Ziv compression scheme to the reinforcement learning setting and assumes n-Markov models of environments; and MC-AIXI-CTW [28], which uses a Bayesian mixture of context trees and incorporates both the Context Tree Weighting algorithm [31] as well as UCT Monte Carlo planning [16]. These can all be viewed as attempts at resolving perceptual aliasing problems with the help of short-term memory. This has turned out to be a more tractable approach than Baum-Welch methods for learning a Partially Observable Markov Decision Process (POMDP) [4] or Predictive State Representations [24]. The history based methods attempt to directly learn the environment states, thereby avoiding the POMDP-learning problem [15], [20] which is extremely hard to solve. Model minimization [8] is a line of works that also seek for a minimal representation of the state space, but focus on solving Markovian problems while ΦMDP and other aforementioned history-based methods target non-Markovian ones. It is also worthy to note that there are various other attempts to find compact representations of MDP state spaces [18]; most of which, unlike our approach, address the planning problem where the MDP model is given

Paper Organization. The paper is organized as follows. Section 2 introduces preliminaries on Reinforcement Learning, Markov Decision Processes, Stochastic Search methods and Context Trees. These are the components from which the ΦMDP algorithm (GSΦA) is built. In Section 3 we put all of the components into our ΦMDP algorithm and also describe our specialized search proposal distribution in detail. Section 4 presents experimental results on four domains. Finally Section 5 summarizes the main results of this paper, and briefly suggests possible research directions.

2 Preliminaries

2.1 Markov Decision Processes (MDP)

An environment is a process which at any discrete time t, given action $a_t \in \mathcal{A}$ produces an observation $o_t \in \mathcal{O}$ and a corresponding reward $r_t \in \mathbb{R}$. When the

process is a Markov Decision Process [27]; o_t represents the environment state, and hence is denoted by s_t instead. Formally, a finite MDP is denoted by a quadruple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ in which $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $T = (T^a_{ss'} : s, s' \in \mathcal{S},\ a \in \mathcal{A})$ 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 $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$, where γ is the geometric discount factor ($0 \leq \gamma < 1$).

Similarly, 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, a useful way to find an estimate of the optimal action values Q^* is to employ the Action-Value Iteration (AVI) algorithm, which is based on the optimal action-value Bellman equation [27], and iterates the update $Q(s, a) \leftarrow \sum_{s'} T^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q(s', a')]$.

If the MDP model is unknown, an effective estimation technique is provided by Q-learning, which incrementally updates estimates Q_t through the equation

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) err_t$$

where the feedback error $err_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$, and $\alpha_t(s_t, a_t)$ is the learning rate at time t. Under the assumption of sufficient visits of all state-action pairs, Q-Learning converges if and only if some conditions of the learning rates are met [2], [27]. In practice a small constant value of the learning rates ($\alpha(s_t, a_t) = \eta$) is, however, often adequate to get a good estimate of Q^*. Q-Learning is off-policy; it directly approximates Q^* regardless of what actions are actually taken. This approach is particularly beneficial when handling the exploration-exploitation tradeoff in RL.

It is well known that learning by taking greedy actions retrieved from the current estimate $\widehat{Q}$ of Q^* to explore the state-action space generally leads to suboptimal behavior. The simplest remedy for this inefficiency is to employ the ϵ-greedy scheme, where with probability $\epsilon > 0$ we take a random action, and with probability $1-\epsilon$ the greedy action is selected. This method is simple, but has shown to fail to properly resolve the exploration-exploitation tradeoff. A more systematic strategy for exploring the unseen scenarios, instead of just taking random actions, is to use optimistic initial values [27], [3]. To apply this idea to Q-Learning, we simply initialize $Q(s, a)$ with large values. Suppose $R_{\max}$ is the maximal reward, Q initializations of at least $\frac{R_{\max}}{1-\gamma}$ are optimistic as $Q(s, a) \leq \frac{R_{\max}}{1-\gamma}$.

2.2 Feature Reinforcement Learning

Problem description. An RL agent aims to find the optimal policy π for taking action a_t given the history of past observations, rewards and actions $h_t = o_1 r_1 a_1 \ldots o_{t-1} r_{t-1} a_{t-1} o_t r_t$ in order to maximize the long-term reward signal. If the problem satisfies an MDP; as can be seen above, efficient solutions are available. We

aim to attack the most challenging RL problem where the environment's states and model are both unknown. In [13], this problem is named the Universal Artificial Intelligence (AI) problem since almost all AI problems can be reduced to it.

ΦMDP framework. In [14], Hutter proposes a history-based method, a general statistical and information theoretic framework called ΦMDP. This approach offers a critical preliminary reduction step to facilitate the agent's ultimate search for the optimal policy. The general ΦMDP framework endeavors to extract relevant features for reward prediction from the past history h_t by using a feature map Φ: $\mathcal{H} \to \mathcal{S}$, where $\mathcal{H}$ is the set of all finite histories. More specifically, we want the states $s_t = \Phi(h_t)$ and the resulting tuple $\langle \mathcal{S},\ \mathcal{A},\ \mathcal{R} \rangle$ to satisfy the Markov property of an MDP. As aforementioned, one of the most useful classes of Φs is the class of context trees, where each tree maps a history to a single state represented by the tree itself. A more general class of Φ is Probabilistic-Deterministic Finite Automata (PDFA) [29], which map histories to the MDP states where the next state can be determined from the current state and the next observation. The primary purpose of ΦMDP is to find a map Φ so that rewards of the MDP induced from the map can be predicted well. This enables us to use MDP solvers, like AVI and Q-learning, on the induced MDP to find a good policy. The reduction quality of each Φ is dictated by the capability of predicting rewards of the resulting MDP induced from that Φ. A suitable cost function that measures the utility of Φs for this purpose is essential, and the optimal Φ is the one that minimizes this cost function.

Cost function. The cost used in this paper is an extended version of the original cost introduced in [14]. We define a cost that measures the reward predictability of each Φ, or more specifically of the resulting MDP induced from that Φ. Based on this, our cost includes the description length of rewards; however, rewards depend on states as well, so the description length of states must be also added to the cost. In other words, the cost comprises coding of the rewards and resulting states, and is defined as follows:

$$\mathbf{Cost}_\alpha(\Phi|h_n) := \alpha \mathbf{CL}(s_{1:n}|a_{1:n}) + (1-\alpha)\mathbf{CL}(r_{1:n}|s_{1:n}, a_{1:n})$$

where $s_{1:n} = s_1, ..., s_n$ and $a_{1:n} = a_1, ..., a_n$ and $s_t = \Phi(h_t)$ and $h_t = ora_{1:t-1}r_t$ and $0 \le \alpha \le 1$. For coding we use the two-part code [30], [10], hence the code length (CL) is $\mathbf{CL}(x) = \mathbf{CL}(x|\theta) + \mathbf{CL}(\theta)$ where x denotes the data sampled from the model specified by parameters θ. We employ the optimal codes [5] for describing data $\mathbf{CL}(x|\theta) = \log(1/Pr_\theta(x))$, while parameters are uniformly encoded to precision $1/\sqrt{\ell(x)}$ where $\ell(x)$ is the sequence length of x [10]: $\mathbf{CL}(\theta) = \frac{m-1}{2}\log \ell(x)$, here m is the number of parameters. The optimal Φ is found via the optimization problem $\Phi^{optimal} = \text{argmin}_\Phi\, \mathbf{Cost}_\alpha(\Phi|h_n)$.

Denote $\mathbf{n}_\bullet := [n_1\ n_2 \dots\ n_l]$ (l is determined in specific context); $n_+ := \sum_j n_j$ (n_js are components of vector $\mathbf{n}_\bullet$); $|\bullet|$ cardinality of a set; $n_{ss'}^{ar'} := |\{t : (s_t, a_t, s_{t+1}, r_{t+1}) = (s, a, s', r'),\ 1 \le t \le n\}|$; and $\mathbf{H}(\mathbf{p}) = -\sum_{i=1}^{l} p_i \log p_i$ Shannon entropy of a random

variable with distribution $\mathbf{p} = [p_1\ p_2 \ldots\ p_l]$ where $\sum_{i=1}^{l} p_i = 1$. The state and reward cost functions can, then, be analytically computed as follows:

$$\mathbf{CL}(s_{1:n}|a_{1:n}) = \sum_{s,a} \mathbf{CL}(\mathbf{n}_{s\bullet}^{a+}) = \sum_{s,a} n_{s+}^{a+} \mathbf{H}\left(\frac{\mathbf{n}_{s\bullet}^{a+}}{n_{s+}^{a+}}\right) + \frac{|\mathcal{S}|-1}{2}\log n_{s+}^{a+}$$

$$\mathbf{CL}(r_{1:n}|s_{1:n}, a_{1:n}) = \sum_{s,a,s'} \mathbf{CL}(\mathbf{n}_{ss'}^{a\bullet}) = \sum_{s,a,s'} n_{ss'}^{a+} \mathbf{H}\left(\frac{\mathbf{n}_{ss'}^{a\bullet}}{n_{ss'}^{a+}}\right) + \frac{|\mathcal{R}|-1}{2}\log n_{ss'}^{a+}$$

As we primarily want to find a Φ that has the best reward predictability, the introduction of α is primarily to stress on reward coding, making costs for high-quality Φs much lower with very small α values. In other words, α amplifies the differences among high-quality Φs and bad ones; and this accelerates our stochastic search process described below.

We furthermore replace $\mathbf{CL}(x)$ with $\mathbf{CL}_\beta(x) = \mathbf{CL}(x|\theta) + \beta\mathbf{CL}(\theta)$ in $\mathbf{Cost}_\alpha$ to define $\mathbf{Cost}_{\alpha,\beta}$ for the purpose of being able to select the right model given limited data. The motivation to introduce β is the following. For stationary environments the cost function is analytically of this form $C_1 \times u(\alpha) \times O(n) + C_2 \times v(\alpha) \times t(\beta) \times O(\log(n))$ where C_1, C_2 are constants, and u, v, t are linear functions. The optimal Φ should be the one with the smallest value of $C_1 \times u(\alpha)$, however, the curse here is that in practice $C_2 \times v(\alpha)$ is often big, so in order to obtain the optimal Φ with limited data, a small value of β will help. We assert that with a very large number of samples n, α and β can be ignored in the above cost function (use $\alpha = 0.5,\ \beta = 1$ as the cost in [14]). The choice of small α and β helps us more quickly to overcome the model penalty and find the optimal map. This strategy is a quite common practice in statistics, and even in the Minimum Description Length (MDL) community [10]. For instance, AIC [1] uses a very small $\beta = 2/\log n$.

The interested reader is referred to [14] for more detailed analytical formulas, and [26] for further motivation and consistency proofs of the ΦMDP model.

2.3 Context Trees

The class of maps that we will base our algorithm on is a class of context trees.

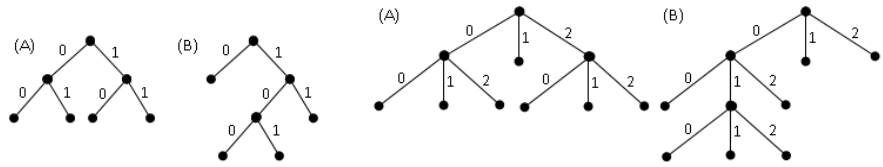
Observation Context Tree (OCT). OCT is a class of maps Φ used to extract relevant information from histories that include only past observations, not actions and rewards. The presentation of OCT is mainly to facilitate the definitions of the below Action-Observation Context Tree.

Definition. Given an $|\mathcal{O}|$-ary alphabet $\mathcal{O} = \{o^1, o^2, \ldots, o^{|\mathcal{O}|}\}$, an OCT constructed from the alphabet $\mathcal{O}$ is defined as a $|\mathcal{O}|$-ary tree in which edges coming from any internal node are labeled by letters in $\mathcal{O}$ from left to right in the order given.

Given an OCT $\mathcal{T}$ constructed from the alphabet $\mathcal{O}$, the state suffix set, or briefly state set $\mathcal{S} = \{s^1, s^2, \ldots, s^m\} \subseteq \mathcal{O}^*$ induced from $\mathcal{T}$ is defined as the set of all possible strings of edge labels forming along a path from a leaf node to the root

node of $\mathcal{T}$. $\mathcal{T}$ is called a Markov tree if it has the so-called Markov property for its associated state set, that is, for every $s^i \in \mathcal{S}$ and $o^k \in \mathcal{O}$, $s^i o^k$ has a unique suffix $s^j \in \mathcal{S}$. The state set of a Markov OCT is called Markov state set. OCTs that do not have the Markov property are identified as non-Markov OCTs. Non-Markov state sets are similarly defined.

Example. Figure 1(a)(A) and 1(a)(B) respectively represent two binary OCTs of depths two and three; also Figures 1(b)(A) and 1(b)(B) illustrate two ternary OCTs of depths two and three.



(a) **Binary context trees** (b) **Trinary context trees**

Figure 1: **Context Trees**

As can be seen from Figure 1, trees 1(a)(A) and 1(b)(A) are Markov; on the other hand, trees 1(a)(B) and 1(b)(B) are non-Markov. The state set of tree 1(a)(A) is $\mathcal{S}^{(a)(A)} = \{00, 01, 01, 11\}$; and furthermore with any further observation $o \in \mathcal{O}$ and $s \in \mathcal{S}^{(a)(A)}$, there exists a unique $s' \in \mathcal{S}$ which is a suffix of so. Hence, tree 1(a)(A) is Markov. Table 1(a) represents the deterministic relation between s, o and s'.

(a) **Markov property of $\mathcal{S}^{(a)(A)}$**

s	00	01	10	11	00	01	10	11
o	0				1			
s'	00	10	00	10	01	11	01	11

(b) **Non-markov property of $\mathcal{S}^{(a)(B)}$**

s	0	001	101	11	0	001	101	11
o	0				1			
s'	0	0	0	0	101 or 001	11	11	11

Table 1: **Markov and Non-Markov properties**

However, there is no such relation in tree 1(a)(B), or state set $\mathcal{S}^{(a)(B)} = \{0, 001, 101, 11\}$; for $s = 0$ and $o = 1$, it is ambiguous whether s' =101 or 001. Table 1(b) clarifies the non-Markov property of tree 1(a)(B).

Similar arguments can be applied for trees 1(b)(A) and 1(b)(B) to identify their Markov property.

It is also worthy to illustrate how an OCT can be used as a map. We illustrate the mapping using again the OCTs in Figure 1. Given two histories including only

past observations $h_5 = 11101$ and $h'_6 = 211210$, then $\Phi^{(a)(A)}(h_5) = 01, \Phi^{(a)(B)}(h_5) = 101, \Phi^{(b)(A)}(h'_6) = 10,$ and $\Phi^{(b)(B)}(h'_6) = 210$.

Action-Observation Context Tree (AOCT). AOCTs are extended from the OCTs presented above for the generic RL problem where relevant histories contain both actions and observations.

Definition. Given two alphabets, $\mathcal{O} = \{o^1, o^2, \ldots, o^{|\mathcal{O}|}\}$ named observation set, and $\mathcal{A} = \{a^1, a^2, \ldots, a^{|\mathcal{A}|}\}$ named action set, an AOCT constructed from the two alphabets is defined as a tree where any internal node at even depths has branching factor $|\mathcal{O}|$, and edges coming from such nodes are labeled by letters in $\mathcal{O}$ from left to right in the order given; and similarly any internal node at odd depths has branching factor $|\mathcal{A}|$, and edges coming from these nodes are labeled by letters in $\mathcal{A}$ also from left to right in the specified order.

The definitions of Markov and non-Markov AOCTs are similar to those of OCTs except that a next observation is now replaced by the next action and observation. Formally, suppose $\mathcal{T}$ is an AOCT constructed from the above two alphabets; and $\mathcal{S} = \{s^1, s^2, \ldots, s^m\} \subseteq (\mathcal{A} \times \mathcal{O})^* \cup \mathcal{A} \times (\mathcal{A} \times \mathcal{O})^*$ is the state suffix set of the tree, then $\mathcal{T}$ is defined as a Markov AOCT if it has the Markov property, that is, for every $1 \leq i \leq m$, $1 \leq j \leq |\mathcal{A}|$, and $1 \leq k \leq |\mathcal{O}|$ there exist a unique $1 \leq l \leq m$ such that s^l is a suffix of $s^i a^j o^k$. AOCTs that do not have Markov property are categorized as non-Markov AOCTs.

The total number of AOCTs up to a certain depth d, $K(d)$, can be recursively computed via the formula $K(d{+}2) = \{[K(d)]^{|\mathcal{A}|}{+}1\}^{|\mathcal{O}|}{+}1$ where $K(0) = 1, K(1) = 2$. As can be easily seen from the recursive formula, the total number of AOCTs is doubly exponential in the tree depth.

An important point to note here is that in our four experiments presented in Section 4, the Φ space is limited to Markov AOCTs, since as explained above, the state suffix set induced from a non-Markov AOCT does not represent an MDP state set; to put it more clearly, in non-Markov AOCTs, from the next action and observation, we cannot derive the next state from the current one. The Markov constraint on AOCTs significantly reduces the search space for our stochastic search algorithm. In the U-tree algorithm [21], no distinction of Marov and non-Markov trees is identified; the algorithm attempts to search for the optimal tree over the whole space of AOCTs.

2.4 Stochastic search

While we have defined the cost criterion for evaluating maps, the problem of finding the optimal map remains. When the Φ space is huge, e.g. context-tree map space where the number of Φs grows doubly exponentially with the tree depth, exhaustive search is unable to deal with domains where the optimal Φ is non-trivial. Stochastic search is a powerful tool for solving optimization problems where the landscape of the objective function is complex, and it appears impossible to analytically or

numerically find the exact or even approximate global optimal solution. A typical stochastic search algorithm starts with a predefined or arbitrary configuration (initial argument of the objective function or state of a system), and from this generates a sequence of configurations based on some predefined probabilistic criterion; the configuration with the best objective value will be retained. There are a wide range of stochastic search methods proposed in the literature [23]; the most popular among these are simulated-annealing-type algorithms [19], [25]. An essential element of a simulated-annealing (SA) algorithm is a Markov Chain Monte Carlo (MCMC) sampling scheme where a proposed new configuration $\tilde{y}$ is drawn from a proposal distribution $q(\tilde{y}|y)$, and we then change from configuration y to $\tilde{y}$ with probability $\min\{1, \frac{\pi_T(y)q(y|\tilde{y})}{\pi_T(\tilde{y})q(\tilde{y}|y)}\}$ where π_T is a target distribution. In a simulated-annealing (SA) algorithm where the traditional Metropolis-Hasting sampling scheme is utilized, π_T is proportional to $e^{-f(x)/T}$ if f is an objective function that we want to minimize, and T is some positive constant temperature. $\frac{q(y|\tilde{y})}{q(\tilde{y}|y)}$ is called the correction factor; it is there to compensate for bias in q.

The traditional SA uses an MCMC scheme with some temperature-decreasing strategy. Although shown to be able to find the global optimum asymptotically [9], it generally works badly in practice as we do not know which temperature cooling scheme is appropriate for the problem under consideration. Fortunately in the ΦMDP cost function we know typical cost differences between two Φs ($C\beta \times \log(n)$), so the range of appropriate temperatures can be significantly reduced. The search process may be improved if we run a number of SA procedures with various different temperatures. Parallel Tempering (PT) [7], [12], an interesting variant of the traditional SA, significantly improves this stochastic search process by smartly offering a swapping step, letting the search procedure use small temperatures for exploitation and big ones for exploration.

Parallel tempering. PT performs stochastic search over the product space $\mathcal{X}_1 \times \ldots \times \mathcal{X}_I (\mathcal{X}_i = \mathcal{X}\ \forall 1 \leq i \leq I)$, where $\mathcal{X}$ is the objective function's domain, and I is the parallel factor. Fixed temperatures T_i ($i = 1, \ldots, I$, and $1 < T_1 < T_2 < \ldots < T_I$) are chosen for spaces $\mathcal{X}_i$ ($i = 1, \ldots, I$). Temperatures T_i ($i = 1, \ldots, I$) are selected based on the following formula $(\frac{1}{T_i} - \frac{1}{T_{i+1}})|\Delta H| \approx -\log p_a$ where ΔH is the "typical" difference between function values of two successive configurations; and p_a is the lower bound for the swapping acceptance rate. The main steps of each PT loop are as follows:

- $(x_1^{(t)}, \ldots, x_I^{(t)})$ is the current sampling; draw $u \sim$ Uniform[0,1]
- If $u \leq \alpha_0$, update every $x_i^{(t)}$ to $x_i^{(t+1)}$ via some Markov Chain Monte Carlo (MCMC) scheme like Metropolis-Hasting (Parallel step)
- If $u > \alpha_0$, randomly choose a neighbor pair, say i and $i+1$, and accept the swap of $x_i^{(t)}$ and $x_{i+1}^{(t)}$ with probability $\min\{1, \frac{\pi_{T_i}(x_{i+1}^{(t)})\pi_{T_{i+1}}(x_i^{(t)})}{\pi_{T_i}(x_i^{(t)})\pi_{T_{i+1}}(x_{i+1}^{(t)})}\}$ (Swapping step).

The full details of PT are given in Algorithm 1.

Algorithm 1 Parallel Tempering (PT)

Require: An objective function $h(x)$ to be minimized, or equivalently the target distribution $\pi_C \ \alpha \ e^{-h(x)/C}$ for some positive constant C

Require: Swap probability parameter α_0

Require: A proposal distribution $q(y|x)$

Require: Temperatures $T_1, T_2, \ldots, T_L$, and number of iterations N

Initialize arbitrary configurations $x^{(1,1)}, ..., x^{(L,1)}$($\{x^{(k,i)}$: represents the ith value of x for temperature T_k;$\}$)
$x_{opt} \leftarrow \text{argmin}_{x=x^{(\cdot,1)}} \, h(x)$
for $i = 1$ to N **do**
  for $k = 1$ to L **do**
    $\widetilde{y} \leftarrow x^{(k,i-1)}$
    Sample y from the proposal distribution $q(y|\widetilde{y})$
    $r \leftarrow \min\{1, \frac{\pi_{T_k}(y)q(y|\widetilde{y})}{\pi_{T_k}(\widetilde{y})q(\widetilde{y}|y)}\}$ (Metropolis Hastings)
    Draw $u \sim$ Uniform[0,1] and update
    if $u \leq r(\widetilde{y}, y)$ **then**
      $x^{(k,i)} \leftarrow y$
    else
      $x^{(k,i)} \leftarrow \widetilde{y}$
    end if
    if $h(x_{opt}) > h(x^{(k,i)})$ **then**
      $x_{opt} \leftarrow x^{(k,i)}$
    end if
  end for
  Draw $u \sim$ Uniform[0,1]
  if $u \geq \alpha_0$ **then**
    Draw a Uniform $\{1, ..., L-1\}$ and let $b = a + 1$
    $r \leftarrow \min\{1, \frac{\pi_{T_a}(x^{(b,i)})\pi_{T_b}(x^{(a,i)})}{\pi_{T_a}(x^{(a,i)})\pi_{T_b}(x^{(b,i)})}\}$
    Draw $v \sim$ Uniform[0,1]
    if $v \leq r$ **then**
      Swap $x^{(a,i)}$ and $x^{(b,i)}$
    end if
  end if
end for

Return x_{opt}

If its swapping phase is excluded, PT is simply the combination of a fixed number of Metropolis-Hastings procedures. The central point that makes PT powerful is its swapping step where adjacent temperatures interchange their sampling regions. This means that a good configuration can be allowed to use a cooler temperature and exploit what it has found while a worse configuration is given a higher temperature which results in more exploration.

3 The ΦMDP Algorithm

We now describe how the generic ΦMDP algorithm works. The general algorithm is shown below (Algorithm 2). It first takes a number of random actions (5000 in all our experiments). Then it defines the cost function $Cost_{\alpha,\beta}$ based on this history. Stochastic search is then used to find a map Φ with low cost. Based on the optimal Φ the history is transformed into a sequence of states, actions and rewards. We use optimistic frequency estimates from this history to estimate probability parameters for state transitions and rewards. More precisely, we use $\frac{R_{\max}+r_1+...+r_m}{m+1}$ instead of the average $\frac{r_1+...+r_m}{m}$ to estimate expected reward, where $r_1, ..., r_m$ are the rewards that have been observed for a certain state-action pair, and $R_{\max}$ is the highest possible reward. The statistics are used to estimate Q values using AVI. After this the agent starts to interact with the environment again using Q-learning initialized with the values that resulted from the performed AVI. The switch from AVI to Q-Learning is rather obvious, as Q-Learning only needs one cheap update per time step, while AVI requires updating the whole environment model and running a number of value iterations. The first set of random actions might not be sufficient to characterize what the best maps Φ look like, so it might be beneficial to add the new history gathered by the Q-Learning interactions with the environment to the old history, and then repeat the process but without the initial sampling.

Algorithm 2 Generic Stochastic ΦMDP Agent (GSΦA)

Require: $Environment$; $initialSampleNumber$, $agentLearningLoops$, $stochasticIterations$ and $additionalSampleNumber$

Generate a history $h^{initial}$ of length $initialSampleNumber$
$h \leftarrow h^{initial}$
repeat
  Run the chosen stochastic search scheme for the history h to find a $\hat{\Phi}$ with low cost
  Compute MDP statistics (optimistic frequency estimates $\hat{R}$ and $\hat{T}$) induced from $\hat{\Phi}$
  Apply AVI to find the optimal Q^* values using the computed statistics $\hat{R}$ and $\hat{T}$.
  Interact with environment for $additionalSampleNumber$ iterations of Q-Learning using Q^* as initial values; the obtained additional history is stored in $h^{additional}$
  $h \leftarrow [h, h^{additional}]$
  $agentLearningLoops \leftarrow agentLearningLoops - 1$
until $agentLearningLoops = 0$
Compute the optimal policy $\pi^{optimal}$ from the optimal Φ and Q values

Return $[\Phi^{optimal}, \pi^{optimal}]$

In the first four experiments in Section 4, PT is employed to search over the Φ

space of Markov AOCTs.

3.1 Proposal Distribution for Stochastic Search over the Markov-AOCT Space

The principal optional component of the above high-level algorithm, GSΦA, is a stochastic search procedure of which some algorithms have been presented in Section 2.4. In these algorithms, an essential technical detail is the proposal distribution q. It is natural to generate the next tree (the next proposal or configuration) from the current tree by splitting or merging nodes. It is possible to express the exact form of our proposal distribution, and based on this to explain how the next tree (next configuration) is proposed from the current tree (current configuration). However, the analytical form of the distribution is cumbersome to specify, so for better exposition we opt to describe the exact behavior of the tree proposal distribution instead.

The stochastic search procedure starts with a Markov AOCT where all of the tree nodes are mergeable, and splittable. However, in the course of the search, a tree node might become unmergeable, but not the other way round; and a splittable node might turn to be unsplittable and vice versa. These specific transfering scenarios are described as follows. A mergeable tree node of the current tree becomes unmergeable if the current tree is proposed from the previous tree by splitting that node, and the cost of the current tree is smaller than that of the previous tree. A splittable leaf node of the current tree becomes unsplittable if the state associated with that node is not present in the current history; however, an unsplittable leaf node might revert to splittable when the state associated with that node is present in the future updated history. The constraint on merging is to keep good short-term memory for predicting rewards, while the other on splitting is simply following the Occam's razor principle.

Merge and split permits. Given some current tree at a particular point in time of the stochastic search process, when considering the generation of the next tree proposal, most of the tree nodes, though labeled splittable and/or mergeable, might have no split, or merge permit, or neither. A node has split permit if it is a leaf node with splittable label. When a leaf node has been split, we simply add all possible children for this node, and label the edges according to the definition of AOCTs. As mentioned above, the newly added leaf nodes might be labeled unmergeable if the cost of the new tree is smaller than that of the old one; and these nodes might also be labeled unsplittable if the states associated with the new leaf nodes are not present in the current history. A node has merge permit if it is labeled mergeable, and all of its children are leaf nodes. When a tree node is merged, all the edges and nodes associated with its children are removed.

Markov-merge and Markov-split permits. Since our search space is the class of Markov OACTs, whenever a split or merge occurs, extra adjustments might be

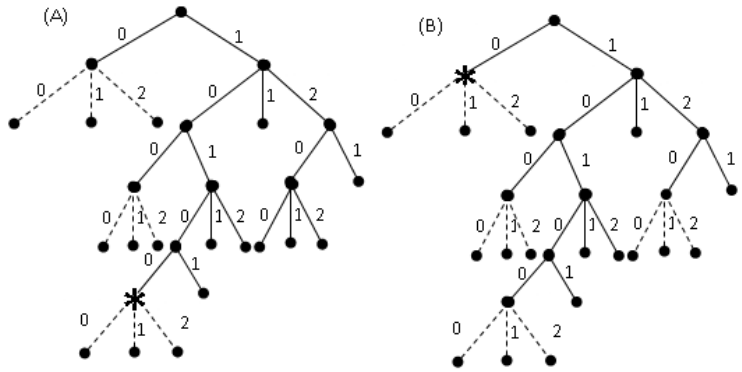


Figure 2: **AOCT proposals**

needed to make the new tree Markov. After a split, there might be nodes that make the tree violate the Markov assumption, and therefore, need to be split. After we split all of those we have to check again to see if any other nodes now need to be split. This goes on until we have a Markov AOCT again. The same applies to merging.

When a node is Markov-split, it and all of the leaf nodes that need to be split (including recursive splits) as a consequence in order to make the tree Markov, are split. A tree node is said to have Markov-split permit if it, and all the other nodes that would be split in a Markov-split of the node, have split permits. This notion is best illustrated with an example. First we define Markov and Non-Markov states of an AOCT. A state of an AOCT is Markov if given any next action-observation pair, the next state is determined; otherwise it is labeled as non-Markov. Now in Figure 2(A), suppose the current Markov AOCT is the tree without dashed edges. Then after splitting the leaf node marked by * (the node associated with state 00101), the state 001 becomes non-Markov so this associated node needs to be split. However, after splitting this node (node associated with state 001), state 0 becomes non-Markov, hence it needs splitting as well. In short, to split the node marked by *, the two nodes associated with states 001 and 0 have to be split as well so as to ensure the resulting tree is Markov after splitting. Similarly, a tree node has Markov-merge permit if it, and all of the tree nodes that minimally and recursively need to be merged after the original node is merged in order to make the tree Markov, have merge permits. For example, in Figure 2(B), suppose the current tree is the tree including both solid and dashed edges, then the node marked by * has Markov-merge permit, if it itself, and the nodes associated with paths 001, 021 and 00101 that need to be merged, have merge permits. When a node with Markov-merge permit is Markov-merged, it and its Markov-merge-associated nodes are merged.

Our procedure to generate the next tree from the current tree (draw sample from $q(y|\cdot)$) in the space of Markov AOCTs consists of the following main steps:

- From the given tree, identify two sets: one is N_S containing nodes with

Markov-split permits, and the other N_M containing nodes with Markov-merge permits.

- Suppose that either N_S or N_M is non-empty otherwise the algorithm (GSΦA) must stop; then if either N_S or N_M is empty, select a node uniformly at random from the other set; otherwise select N_S or N_M randomly with probability $\frac{1}{2}$ each, and after that choose a tree node randomly from the selected set.

- Markov-split the node if it belongs to N_S, otherwise Markov-merge it

Once we have drawn the new tree $\tilde{y}$, the Metropolis Hastings correction factor can be straightforwardly calculated via the formula

$$\frac{q(y|\widetilde{y})}{q(\widetilde{y}|y)} = \begin{cases} \frac{|\widetilde{N}_M|}{|N_S|} & \text{if } \widetilde{y} \text{ is proposed from } y \text{ by Markov-splitting} \\ \frac{|\widetilde{N}_S|}{|N_M|} & \text{if } \widetilde{y} \text{ is proposed from } y \text{ by Markov-merging} \end{cases}$$

here $\widetilde{N}_S$ and $\widetilde{N}_M$ are respectively the set of nodes with Markov-split permits, and the set of nodes with Markov-merge permits of $\widetilde{y}$.

Sharing. If the stochastic search algorithm utilized is PT, we apply another trick to effectively accelerate the search process. Whenever a node is labeled unmergeable, that is, by splitting this node the cost function decreases, or in other words a good additional relevant short-term memory for predicting rewards is found, the states associated with the new nodes created by the splitting are replicated in the trees with the other temperatures.

4 Experiments

4.1 Experimental Setup

Below in this section we present our empirical studies of the ΦMDP algorithm GSΦA described in Section 3. For all of our experiments, stochastic search (PT) is applied in the Φ space of Markov AOCTs.

For a variety of tested domains, our algorithm produces consistent results using the same set of parameters. These parameters are shown in Table 4.1, and are **not** fine tuned.

The results of ΦMDP and the three competitors in the four above-listed environments are shown in Figures 3, 4, 7 and 8. In each of the plots, various time points are chosen to assess and compare the quality of the policies learned by the four approaches. In order to evaluate how good a learned policy is, at each point, the learning process of each agent, and the exploration of the three competitors are temporarily switched off. The selected statistic to compare the quality of learning is the averaged reward over 5000 actions using the current policy. For stability, the statistic is averaged over 10 runs.

Parameter	Component	Value
α	$Cost_{\alpha,\beta}$	0.1
β	$Cost_{\alpha,\beta}$	0.1
$initialSampleNumber$	GSΦA	5000
$agentLearningLoops$	GSΦA	1
Iterations	PT	100
I	PT	10
$T_i,\ i \leq I$	PT	$T_i = \beta \times i \times \log(n)$
α_0	PT	0.7
γ	AVI, Q-Learning	0.999999
η	Q-Learning	0.01

Table 2: **Parameter setting for the GSΦA algorithm**

As shown in more detail below, ΦMDP is superior to U-tree and active-LZ, and is comparable to MC-AIXI-CTW in short-term memory domains. Overall conclusions are clear, and we, therefore, omit error bars.

4.2 Environments and results

We describe each environment, the resulting performance, and the tree that was found by ΦMDP in the cheese maze domain.

4 × 4 Grid. The domain is a 4×4 grid world. At each time step, the agent can

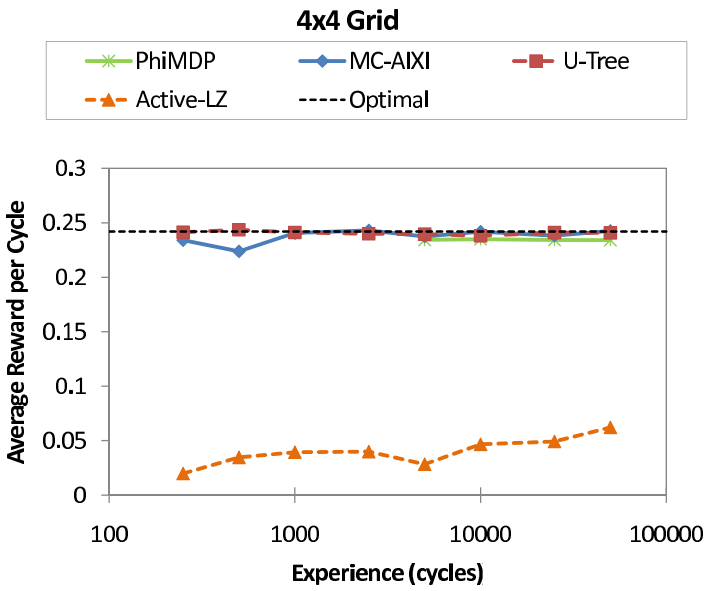


Figure 3: 4×4 **Grid**

move one cell left, right, up and down within the grid world. The observations are uninformative. When the agent enters the bottom-right corner of the grid; it gets a reward of 1, and is automatically and randomly sent back to one of the remaining 15 cells. Entering any cell other than the bottom-right one gives the agent a zero reward. To achieve the maximal total reward, the agent must be able to remember a series of smart actions without any clue about its relative position in the grid.

The context tree found contains 34 states. Some series of actions that take the agent towards the bottom-right corner of the grid are present in the context tree. As shown in the 4×4-grid plot in Figure 3, after 5000 experiences gathered from the random policy, ΦMDP finds the optimal policy, and so does MC-AIXI-CTW and U-Tree. Active-LZ, however, does not converge to an optimal policy even after 50,000 learning cycles.

Tiger. The tiger domain is described as follows. There are two doors, left and right; an amount of gold and a tiger are placed behind the two doors in a random order. The person has three possible actions: listen to predict the position of the tiger, open the right door, and open the left door. If the person listens, he has to pay some money (reward of -1). The probability that the agent hears correctly is 0.85. If the person opens either of the doors and sees the gold, the obtained reward is 10; or otherwise he faces the tiger, then the agent receives a reward of -100. After the door is opened, the episode ends; and in the next episode the tiger sits randomly again behind either the left or the right door.

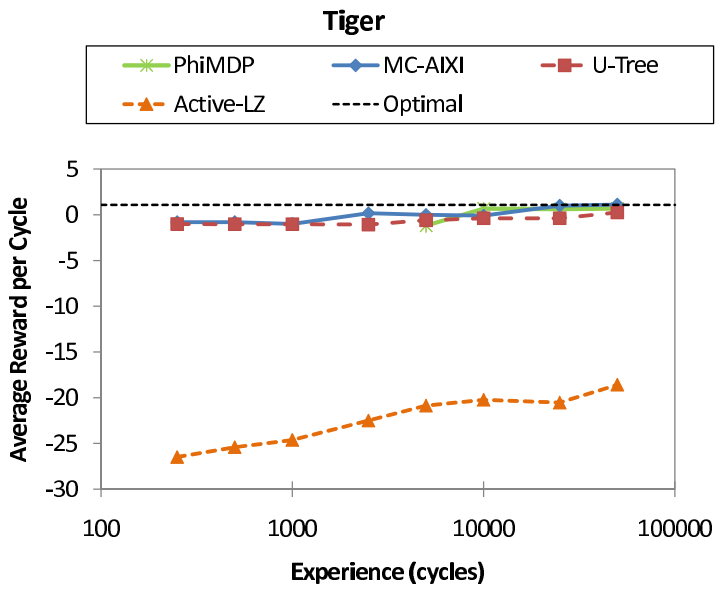


Figure 4: **Tiger**

Our parallel tempering procedure found a context tree consisting of 39 states including some important states where the history is such that the agent has listened a few times before opening the door. It can be seen from the tiger plot in Figure

4 that the optimal policy ΦMDP found after 5,000 learning experiences does yield positive reward on average, while from time point 10,000 on, it achieves as high rewards as MC-AIXI-CTW. U-Tree appears to learn more slowly but eventually manages to get positive averaged rewards after 50,000 cycles like ΦMDP and MC-AIXI-CTW. Active-LZ is performing far worse. The optimal policy that ΦMDP, MC-AIXI-CTW, and U-Tree ultimately found is the following. First listen two times, if the listening outcomes are consistent, open the predicted door with gold behind; otherwise take one more listening action, and based on the majority to open the appropriate door.

Cheese Maze. This domain, as shown in Figure 5, consists of a eleven-cell maze

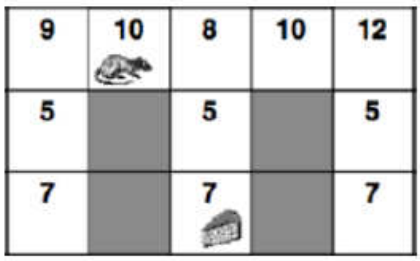


Figure 5: **Cheese-maze domain**

with a cheese in it. The agent is a mouse that attempts to find the cheese. The agent's starting position for each episode is at one of the eleven cells uniformly random. The actions available to the agent are: move one cell left (0), right (1), up (2) and down (3). However, it should be noticed that if the agent hits the wall, its relative position in the maze remains unchanged. At each cell the agent can observe which directions among left, right, up and down the cell is blocked by a wall. If wall-blocking statuses of each cell are represented by 1 (blocked), and 0 (free) respectively; then an observation is described by a four-digit binary number where the digits from left to right are wall-blocking statuses of up, left, down and right directions. For example, 0101 = 5, 0111 = 7, ... as described in Figure 5. The agent gets a reward of -1 when moving into a free cell without a cheese; hitting the wall gives it a penalty of -10; and a reward of 10 is given to the agent when it finds the cheese. As can be seen, some observations themselves alone are insufficient for the mouse to locate itself unambiguously in the maze. Hence, the mouse must learn to resolve these ambiguities of observations in the maze to be able to find the optimal policy.

Our algorithm found a context tree consisting of 43 states that contains the tree as shown in Figure 6. The tree splits from the root into the 6 possible observations. Then observations 5 and 10 are split into the four possible actions; and some of these actions, the ones that come from a different location and not a wall collision, are split further into the 6 "possible" observations before that. This resolves which 5 or

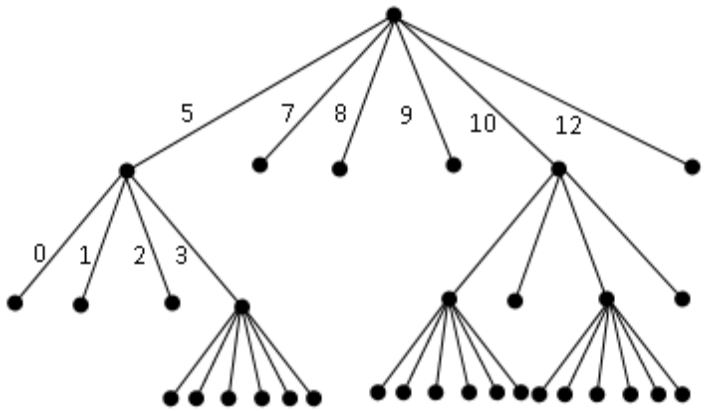


Figure 6: **Cheese-maze tree**

which 10 we are at. The states in this tree resolve the most important ambiguities of the raw observations and an optimal policy can be found. The domain contains an infinite amount of longer dependencies among which our found states pick up a small subset. The cheese-maze plot in Figure 7 shows that after the initial 5000 experiences, ΦMDP is marginally worse than MC-AIXI-CTW but is better than U-Tree and Active-LZ. From time point 10,000, there is no difference between ΦMDP and MC-AIXI-CTW. U-Tree and Active-LZ remain inferior.

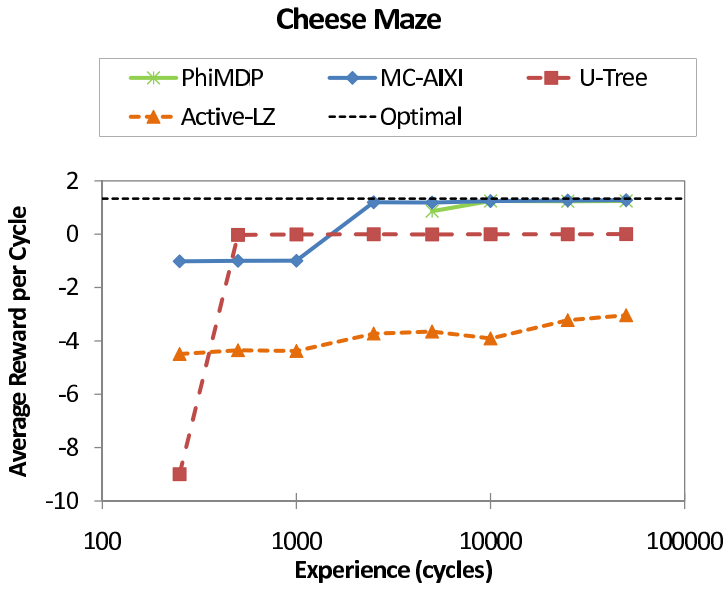


Figure 7: **Cheese maze**

Kuhn Poker. In Kuhn poker [17] a deck of only three cards (Jack, Queen and King) is used. The agent always plays second in any game (episode). After putting

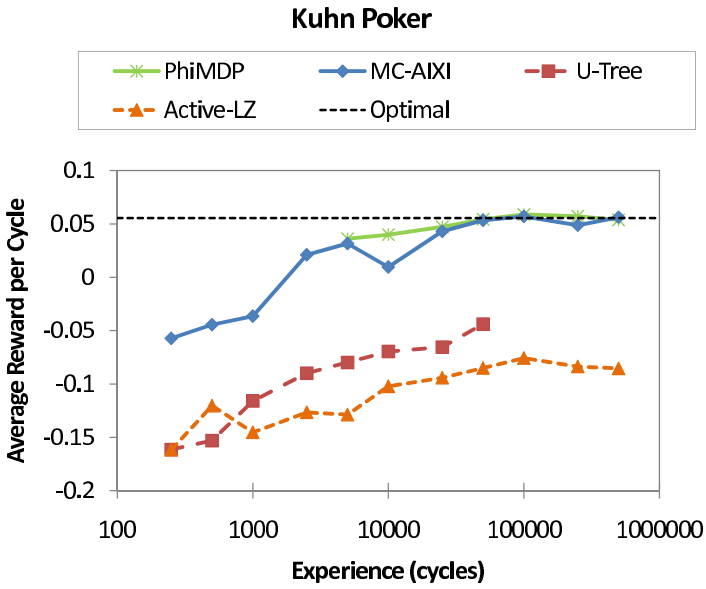


Figure 8: **Kuhn poker**

a chip each into play, the players are dealt a card each. Then the first player says bet or pass and the second player chooses bet or pass. If player one says pass and player two says bet then player one must choose again between bet and pass. Whenever a player says bet they must put in another chip. If one player bets and the other pass the better gets all the chips in play. Otherwise the player with the highest card gets the chips. Player one plays according to a fixed but stochastic Nash optimal strategy [11]. ΦMDP finds 89 states. It can be observed from the Kunh-poker plot in Figure 8 that ΦMDP is comparable to MC-AIXI-CTW and much better than U-Tree and Active-LZ, who loose money.

5 Conclusions

Based on the Feature Reinforcement Learning framework [14] we defined actual practical reinforcement learning agents that perform very well empirically. We evaluated a reasonably simple instantiation of our algorithm that first takes 5000 random actions followed by finding a map through a search procedure and then it performs Q-learning on the MDP defined by the map's state set.

We performed an evaluation on four test domains used to evaluate MC-AIXI-CTW in [28]. Those domains are all suitably attacked with context tree methods. We defined a ΦMDP agent for a class of maps based on context trees, and compared it to three other context tree-based methods. Key to the success of our ΦMDP agent was the development of a suitable stochastic search method for the class of Markov AOCTs. We combined parallel tempering with a specialized proposal distribution that results in an effective stochastic search procedure. The ΦMDP

agent outperforms both the classical U-tree algorithm [21] and the recent Active-LZ algorithm [6], and is competitive with the newest state of the art method MC-AIXI-CTW [28]. The main reason that ΦMDP outperforms U-tree is that ΦMDP uses a global criterion (enabling the use of powerful global optimizers) whereas U-tree uses a local split-merge criterion. ΦMDP also performs significantly better than Active-LZ. Active-LZ learns slowly as it overestimates the environment model (assuming n-Markov or complete context-tree environment models); and this leads to unreliable value-function estimates.

Below are some detailed advantages of ΦMDP over MC-AIXI-CTW:

- ΦMDP is more efficient than MC-AIXI-CTW in both computation and memory usage. ΦMDP only needs an initial number of samples and then it finds the optimal map and uses AVI to find MDP parameters. After this it only needs a Q-learning update for each iteration. On the other hand, MC-AIXI-CTW requires model updating, planning and value-reverting at every single cycle which together are orders of magnitude more expensive than Q-learning. In the experiments ΦMDP finished in minutes while MC-AIXI-CTW needed hours. Another disadvantage of MC-AIXI-CTW is that it is a memory-hungry algorithm. ΦMDP learns the best tree representation using stochastic search, which expands a tree towards relevant histories. MC-AIXI-CTW learns the mixture of trees where the number of tree nodes grows (and thereby the memory usage) linearly with time.

- ΦMDP learns a single state representation and can use many classical RL algorithms, e.g. Q-Learning, for MDP learning and planning.

- Another key benefit is that ΦMDP represents a more discriminative approach than MC-AIXI-CTW since it aims primarily for the ability to predict future rewards and not to fully model the observation sequence. If the observation sequence is very complex, this becomes essential.

On the other hand, to be fair it should be noted that compared to ΦMDP, MC-AIXI-CTW is more principled. The results presented in this paper are encouraging since they show that we can achieve comparable results to the more sophisticated MC-AIXI-CTW algorithm on problems where only short-term memory is needed. We plan to utilize the aforementioned advantages of the ΦMDP framework, like flexibility in environment modeling and computational efficiency, to attack more complex and larger problems.

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 experimental comparison.

References

[1] Akaike, H.: A new look at the statistical model identification. IEEE Transactions on Automatic Control 19, 716–723 (1974)

[2] Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Anthena Scientific, Belmont, MA (1996)

[3] Brafman, R.I., Tennenholz, M.: R-max -a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learing Research 3, 213–231 (2002)

[4] Chrisman, L.: Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In: AAAI. pp. 183–188 (1992)

[5] Cover, T.M., Thomas, J.A.: Elements of Information Theory. John Willey and Sons (1991)

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

[7] Geyer, C.J.: Markov chain Monte Calro maximum likelihood. In: Computing Science and Statistics: the 23rd Symposium on the Interface. pp. 156–163. Interface Foundation, Fairfax (1991)

[8] Givan, R., Dean, T., Greig, M.: Equivalence notions and model minimization in Markov decision process. Artificial Intelligence 147, 163–223 (2003)

[9] Granville, V., Křivánek, M., Rasson, J.P.: Simulated annealing: A proof of convergence. IEEE Transactions on Pattern Analysis and Machine Intelligence 16(6), 652–656 (June 1994)

[10] Grünwald, P.D.: The Minimum Description Length Principle. The MIT Press (2007)

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

[12] Hukushima, K., Nemoto, K.: Exchange monte carlo method and application to spin glass simulations. Journal of the Physical Socieity of Japan 65(4), 1604–1608 (1996)

[13] Hutter, M.: Universal Articial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Berlin (2005)

[14] Hutter, M.: Feature reinforcement learning: Part I. Unstructured MDPs. Journal of General Artificial Intelligence (2009)

[15] Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in paritally observable stochastic domains. Artifical Intelligence 101, 99–134 (1998)

[16] Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: The 17th European Conference on Machine Learning. pp. 99–134 (2006)

[17] Kuhn, H.W.: A simplified two-persion poker. In: Contributions to the Theory of Games. pp. 97–103 (1950)

[18] Li, L., Walsh, T.J., Littmans, M.L.: Towards a unified theory of state abstraction for Mdps. In: In Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics (2006)

[19] Liu, J.S.: Monte Carlo Strategies in Scientific Computing. Springer (2001)

[20] Madani, O., Handks, S., Condon: On the undecidability of probabilistic planning and related stochastic optimization problems. Artifical Intelligence 147, 5–34 (2003)

[21] McCallum, A.K.: Reinforcement Learning with Selective Perception and Hidden State. Ph.D. thesis, Department of Computer Science, University of Rochester (1996)

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

[23] Schneider, J., Kirkpatrick, S.: Stochastic Optimization. Springer, first edn. (2006)

[24] Singh, S.P., James, M.R., Rudary, M.R.: Predictive state representations: A new theory for modeling dynamical systems. In: Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence. pp. 512–518. Banff, Canada (2004)

[25] Suman, B., Kumar, P.: A survey of simulated annealing as a tool for single and multiobjecctive optimization. Journal of the Operational Research Society 57, 1143–1160 (2006)

[26] Sunehag, P., Hutter, M.: Consistency of feature Markov processes. In: Proc. 21st International Conf. on Algorithmic Learning Theory (ALT'10). LNAI, vol. 6331, pp. 360–374. Springer, Berlin, Canberra (2010)

[27] Sutton, R., Barto, A.: Reinforcement Learning. The MIT Press (1998)

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

[29] Vidal, E., Thollard, F., Higuera, C.D.L., Casacuberta, F., Carrasco, R.C.: Probabilitic finite-state machines. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(7), 1013–1025 (July 2005)

[30] Wallace, C.S., Dowe, D.L.: Minimum message length and komogorov complexity. Computer Journal 42(4), 270–283 (1999)

[31] Wilems, F.M.J., Shtarkov, Y.M., Tjalkens, T.J.: The context tree weighting method: Basic properties. IEEE Transactions on Information Theory 41, 653–644 (1995)