Bayesian Learning of Recursively Factored Environments

Marc Bellemare MG17@CS.UALBERTA.CA
Joel Veness VENESS@CS.UALBERTA.CA
Michael Bowling BOWLING@CS.UALBERTA.CA
University of Alberta, Edmonton, Canada, T6G 2E8

Abstract

Model-based reinforcement learning techniques have historically encountered a number of difficulties scaling up to large observation spaces. One promising approach has been to decompose the model learning task into a number of smaller, more manageable sub-problems by factoring the observation space. Typically, many different factorizations are possible, which can make it difficult to select an appropriate factorization without extensive testing. In this paper we introduce the class of recursively decomposable factorizations, and show how exact Bayesian inference can be used to efficiently guarantee predictive performance close to the best factorization in this class. We demonstrate the strength of this approach by presenting a collection of empirical results for 20 different Atari 2600 games.

1. Introduction

The ability of a reinforcement learning agent to accurately model its environment can be leveraged in a number of useful ways. In simulation-based search, UCT (Kocsis & Szepesvári, 2006), POMCP (Silver & Veness, 2010) and FSSS (Walsh et al., 2010) can all be used to produce refined value function estimates tailored to the current situation facing the agent. Within the context of Bayesian reinforcement learning, forward search can provide a principled means to explore the environment (Asmuth & Littman, 2011; Guez et al., 2012). Having access to a model also allows for hybrid techniques such as Dyna (Sutton, 1991; Silver et al., 2008) and TD-Search (Silver et al., 2012), which use a model of the environment to enhance the performance of more traditional model-free reinforcement learning techniques.

Often, the environment dynamics are unknown a priori and model-based agents must learn their model from experience. This has recently motivated the development of a number of promising approaches; of note, Doshi-Velez (2009), Walsh et al. (2010), Veness et al. (2010), Veness et al. (2011), Nguyen et al. (2012), and Guez et al. (2012) have collectively demonstrated that it is feasible to learn good probabilistic models of small but challenging domains containing various degrees of partial observability and stochasticity.

More often than not, however, the evaluation of model learning algorithms is performed over synthetic domains designed to illustrate particular algorithmic challenges. Such domains typically feature intricate environment dynamics over small, constrained observation spaces, making the model learning problem perhaps unnecessarily difficult. In this work, we focus on the reverse situation: domains with high-dimensional observation spaces but fundamentally simple dynamics, such as the Arcade Learning Environment (Bellemare et al., 2013), a collection of reinforcement learning domains based on Atari 2600 games. Our aim in this paper is to provide algorithmic scaffolding that lets us scale existing, successful model learning algorithms to larger observation spaces.

One promising approach for scaling model-based reinforcement learning to larger observation spaces is to consider various kinds of *factored models* (Ross & Pineau, 2008; Poupart, 2008; Diuk et al., 2009). A good choice of factorization can make the model learning task much easier by decomposing it into a number of more manageable sub-problems; the resultant performance then depends on the particular choice of factorization. In this paper we introduce a meta-algorithm that efficiently performs exact Bayesian model averaging over a large class of recursively decomposable factorizations, allowing us to provide strong competitive guarantees with respect to the

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

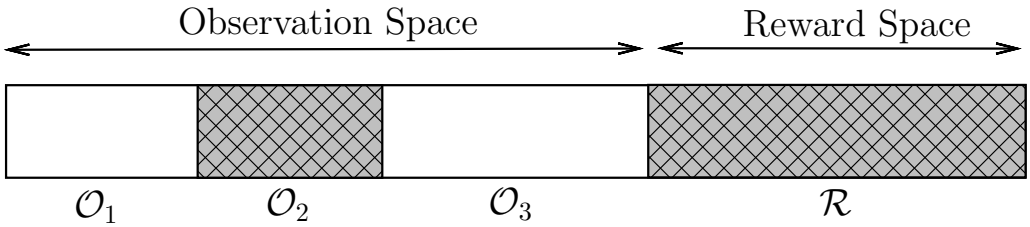


Figure 1. A factored percept space, where $\mathcal{X} = \mathcal{O}_1 \times \mathcal{O}_2 \times \mathcal{O}_3 \times \mathcal{R}$.

best factorization within our class.

Our approach becomes particularly meaningful in the context of developing domain-independent agents, i.e. agents that attempt to perform well over a large set of domains (Bellemare et al., 2013; Hausknecht et al., 2012). In particular, we apply our model averaging techniques to a recursive decomposition of the Atari 2600 image space to obtain efficient and accurate forward models on twenty different games. To the best of our knowledge, our model learning algorithm is the first to successfully handle observation spaces of this magnitude in a model-based reinforcement learning setting.

2. Background

Before we describe our model averaging technique, we first need to introduce some notation for probabilistic agents. The framework used in this paper was introduced by Hutter (2005), and can be seen as a natural generalization of a *coding distribution*, a mathematical object from information theory that is used to assign probabilities to sequential data. This generalization works similarly to how (PO)MDPs (Puterman, 1994; Kaelbling et al., 1998) generalize Markov Models and Hidden Markov Models to the control setting. This general presentation allows us to present our techniques in a way that is applicable to many different kinds of model-based reinforcement learning algorithms, including techniques for discrete (PO)MDPs.

2.1. Probabilistic Agent Setting

This section describes our probabilistic agent setting, how factored models can be specified within it, and how Bayesian model averaging can be used over a class of factored models to predict nearly as well as the best factorization in a given class.

We begin with some notation. A string $x_1 x_2 \dots x_n$ of length n is denoted by $x_{1:n}$. The prefix $x_{1:j}$ of $x_{1:n}$, $j \leq n$, is denoted by $x_{\leq j}$ or $x_{<j+1}$. The notation generalizes to blocks of symbols: e.g. $ax_{1:n}$ denotes $a_1 x_1 a_2 x_2 \dots a_n x_n$ and $ax_{<j}$ denotes the string $a_1 x_1 a_2 x_2 \dots a_{j-1} x_{j-1}$. The empty string is denoted by ϵ. The concatenation of two strings s and r is denoted by sr. The finite action, observation, and reward spaces are denoted by $\mathcal{A}, \mathcal{O}$, and $\mathcal{R}$ respectively. We will also use $\mathcal{X}$ to denote the joint perception space $\mathcal{O} \times \mathcal{R}$.

We now define an environment to be a probability distribution, parametrized by a sequence of actions, over possible observation-reward sequences. More formally, an environment ρ is a sequence of parametrized probability mass functions $\{\rho_0, \rho_1, \rho_2, \dots\}$, where

$$\rho_n \colon \mathcal{A}^n \to (\mathcal{X}^n \to [0,1]),$$

that satisfies, for all $n \in \mathbb{N}$, for all $a_{1:n} \in \mathcal{A}^n$, for all $x_{<n} \in \mathcal{X}^{n-1}$, the constraint that

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

with the base case defined as $\rho_0(\epsilon \,|\, \epsilon) = 1$. Equation 1 captures the intuitive constraint that an action performed at time n should have no effect on earlier perceptions $x_{<n}$. Where possible, from here onwards we will drop the index n in ρ_n when the meaning is clear. Now, given an environment ρ, the predictive probability of a percept x_n can be defined as

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

$\forall a_{1:n} \forall x_{1:n}$ such that $\rho(x_{<n} \,|\, a_{<n}) > 0$. This implies a version of the familiar product rule, i.e.

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

Our notion of environment is used in two distinct ways. The first is as a means of describing the true underlying environment, which may be unknown to the agent. The second is to describe an agent's model of the environment. This model is typically adaptive, and will often only be an approximation to the true environment. To make the distinction clear, we will refer to an agent's *environment model* when talking about the agent's model of the environment. Additionally, we will also introduce the notion of an ϵ-positive environment model. This is defined as an environment model

ρ satisfying $\rho(x_n \mid a x_{<n} a_n) \geq \epsilon$ for some real $\epsilon > 0$, for all $n \in \mathbb{N}$, for all $x_{1:n} \in \mathcal{X}^n$ and for all $a_{1:n} \in \mathcal{A}^n$. From here onwards, we assume that all environment models are ϵ-positive.

2.2. Factored Environment Models

We now introduce some notation for environments whose percept spaces $\mathcal{X}$ can be expressed as $\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_k$, the Cartesian product of $k \in \mathbb{N}$ subspaces. An example of such a factored space is depicted in Figure 1. First, let $\mathcal{X}_{<i} := \mathcal{X}_1 \times \cdots \times \mathcal{X}_{i-1}$ for $1 \leq i \leq k$. This notation will also generalize to n-dimensional string types, with $\mathcal{X}_i^n := (\mathcal{X}_i)^n$ and $\mathcal{X}_{<i}^n := (\mathcal{X}_{<i})^n$. Furthermore, given a string $x_{1:n} \in (\mathcal{X}_1 \times \cdots \times \mathcal{X}_k)^n$, we introduce the notation $x_t^i \in \mathcal{X}_i$ for $1 \leq i \leq k$ and $1 \leq t \leq n$, to denote the ith component of x_t. This will again generalize to the multi-dimensional case, with $x_{1:n}^i := x_1^i x_2^i \ldots x_n^i$.

Now, given an action space $\mathcal{A}$ and a factored percept space $\mathcal{X} := \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$, a k-factored environment is defined by a tuple $(\rho^1, \ldots, \rho^k)$, where each component of the tuple is an environment model factor

$$\rho^i := \left\{ \rho_n^i : (\mathcal{A} \times \mathcal{X})^{n-1} \times \mathcal{A} \times \mathcal{X}_{<i} \to (\mathcal{X}_i \to [0,1]) \right\}_{n \in \mathbb{N}}$$

for $1 \leq i \leq k$, with each ρ_n^i defining a parametrized probability mass function. Using the chain rule, this naturally induces a factored environment given by

$$\begin{aligned} \rho(x_{1:n} \mid a_{1:n}) &= \prod_{t=1}^{n} \rho(x_t \mid a x_{<t}) \\ &= \prod_{t=1}^{n} \prod_{i=1}^{k} \rho_t^i(x_t^i \mid a x_{<t} a_t x_t^{<i}) \\ &= \prod_{i=1}^{k} \rho^i(x_{1:n}^i \mid a_{1:n}), \end{aligned}$$

where the final line uses the notation

$$\rho^i(x_{1:n}^i \mid a_{1:n}) := \prod_{t=1}^{n} \rho_t^i(x_t^i \mid a x_{<t} a_t x_t^{<i}).$$

One can easily verify that a k-factored environment satisfies Equation 1, and is therefore a valid environment.

2.3. Model Averaging over Factorizations

Next we consider a Bayesian approach to learning a factorization online. Given a finite class $\mathcal{M}$ of candidate factorizations, we can define a mixture environment model that averages over the set of factorizations in $\mathcal{M}$ by

$$\xi(x_{1:n} \mid a_{1:n}) := \sum_{\rho \in \mathcal{M}} w_0^\rho \, \rho(x_{1:n} \mid a_{1:n}), \tag{4}$$

where each real weight $w_0^\rho > 0$ and $\sum_{\rho \in \mathcal{M}} w_0^\rho = 1$. This is a direct application of Bayesian model averaging for our kind of probabilistic agent; see (Veness et al., 2011) for a more detailed discussion of this approach. We can show that this method is justified whenever there exists a factorization $\rho^* \in \mathcal{M}$ that predicts well, since

$$\begin{aligned} -\log \xi(x_{1:n} \mid a_{1:n}) &= -\log \sum_{\rho \in \mathcal{M}} w_0^\rho \, \rho(x_{1:n} \mid a_{1:n}) \\ &\leq -\log w_0^{\rho^*} - \log \rho^*(x_{1:n} \mid a_{1:n}), \end{aligned}$$

which implies that we only suffer a constant penalty when using ξ in place of ρ^*. This can be interpreted as a kind of regret bound, where the loss function is given by the code length (Grünwald, 2007), i.e. the number of bits needed to encode the sequence of percepts $x_{1:n}$ under the environment model $\rho(\cdot \mid a_{1:n})$. By taking expectations on both sides with respect to the true environment μ and rearranging, we see that this result implies

$$\mathbb{E}_\mu \left[\log \frac{\rho^*(x_{1:n} \mid a_{1:n})}{\xi(x_{1:n} \mid a_{1:n})} \right] \leq -\log w_0^{\rho^*}.$$

This can be rewritten as

$$\mathbb{E}_\mu \left[\log \frac{\mu(x_{1:n} \mid a_{1:n}) \rho^*(x_{1:n} \mid a_{1:n})}{\xi(x_{1:n} \mid a_{1:n}) \mu(x_{1:n} \mid a_{1:n})} \right] \leq -\log w_0^{\rho^*},$$

which upon rearranging gives

$$D_{1:n}(\mu \,\|\, \xi) \leq -\log w_0^{\rho^*} + D_{1:n}(\mu \,\|\, \rho^*), \tag{5}$$

with $D_{1:n}(\mu \,\|\, \rho) := \sum_{x_{1:n}} \mu(x_{1:n} \mid a_{1:n}) \log \frac{\mu(x_{1:n} \mid a_{1:n})}{\rho(x_{1:n} \mid a_{1:n})}$ denoting the KL divergence between the distributions $\mu(\cdot \mid a_{1:n})$ and $\rho(\cdot \mid a_{1:n})$. Dividing Equation 5 by n highlights the per-step penalty $-\frac{1}{n} \log w_0^{\rho^*}$ of using ξ, rather than ρ^*, to predict $x_{1:n}$. As $w_0^{\rho^*}$ does not depend on n, this penalty vanishes asymptotically. Intuitively, the bound in Equation 5 implies that the environment model ξ is in some sense close, as n gets large, to the environment model that uses the best factorization in $\mathcal{M}$.

The main drawback with Bayesian model averaging is that it can be computationally expensive. Ideally we would like to weight over many possible candidate factorizations, but this typically isn't possible if Equation 4 is implemented directly. Our main contribution in this paper is to show that, by carefully choosing

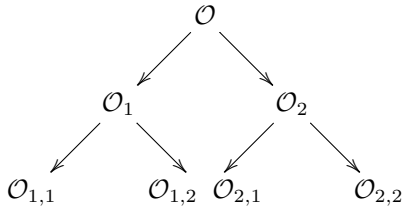
both the prior and the class of possible factorizations, model averaging can be used to derive a practical technique that allows us to build probabilistic agents that do not need to commit to a particular factorization in advance.

3. Recursive Factorizations

We now introduce our efficiently computable class of structured factorizations. Here we focus on a domain-independent presentation and defer to Section 4 a more concrete instantiation tailored to the Atari 2600 platform. We restrict our attention to factoring the observation space, noting that this imposes no restrictions as the reward can always be modelled by a separate environment model factor.

Definition 1. *A recursively decomposable space of nesting depth $d = 0$ is a set. When $d \in \mathbb{N}$, a recursively decomposable space is the set formed from the Cartesian product of two or more recursively decomposable spaces of nesting depth $d - 1$. The number of factors in the Cartesian product defining a recursively decomposable space $\mathcal{F}$ at a particular nesting depth is denoted by $\dim(\mathcal{F})$, and is defined to be 1 if the nesting depth is 0.*

For instance, a recursively decomposable space $\mathcal{O}$ with nesting depth 2 could be defined as: $\mathcal{O} := \mathcal{O}_1 \times \mathcal{O}_2$; $\mathcal{O}_1 := \mathcal{O}_{1,1} \times \mathcal{O}_{1,2}$; $\mathcal{O}_2 := \mathcal{O}_{2,1} \times \mathcal{O}_{2,2}$, with $\mathcal{O}_{1,1}$, $\mathcal{O}_{1,2}$, $\mathcal{O}_{2,1}$ and $\mathcal{O}_{2,2}$ being (arbitrary) sets. The recursive structure in $\mathcal{O}$ has a natural representation as a tree:



where the set appearing at each non-leaf node is formed from the Cartesian product of its children. In this example, $\dim(\mathcal{O}) = \dim(\mathcal{O}_1) = \dim(\mathcal{O}_2) = 2$ and $\dim(\mathcal{O}_{1,1}) = \dim(\mathcal{O}_{1,2}) = \dim(\mathcal{O}_{2,1}) = \dim(\mathcal{O}_{2,2}) = 1$. In what follows, we will continue to use the notation $\mathcal{O}_k$ to denote the kth factor of the Cartesian product defining the recursively decomposable space $\mathcal{O}$. A recursively decomposable space can clearly be factored in many possible ways. In preparation for defining a prior over factorizations, we now define the set of all possible factorizations that can be applied to a particular choice of recursively decomposable space.

Definition 2. *Given a recursively decomposable space $\mathcal{F}$ with nesting depth at least $d \in \mathbb{N}$, the set $\mathcal{C}_d(\mathcal{F})$ of all recursive factorizations of $\mathcal{F}$ is defined by*

$$\mathcal{C}_d(\mathcal{F}) := \{\mathcal{F}\} \cup \bigtimes_{i=1}^{\dim(\mathcal{F})} \mathcal{C}_{d-1}(\mathcal{F}_i), \tag{6}$$

with $\mathcal{C}_0(\mathcal{F}) := \{\mathcal{F}\}$.

Returning to our example, this gives the following set of factorizations: $\mathcal{C}_2(\mathcal{O}) = \{\mathcal{O}, \mathcal{O}_1 \times \mathcal{O}_2, \mathcal{O}_{1,1} \times \mathcal{O}_{1,2} \times \mathcal{O}_2, \quad \mathcal{O}_1 \times \mathcal{O}_{2,1} \times \mathcal{O}_{2,2}, \mathcal{O}_{1,1} \times \mathcal{O}_{1,2} \times \mathcal{O}_{2,1} \times \mathcal{O}_{2,2}\}$. Notice that although the number of factorizations for the above example is small, in general the number of possible factorizations can grow super-exponentially in the nesting depth.

3.1. A Prior over Recursive Factorizations

Now we describe a prior on $\mathcal{C}_d(\mathcal{F})$ that will support efficient computation, as well as being biased towards factorizations containing less nested substructure.

First note that in the recursive construction of $\mathcal{C}_d(\mathcal{F})$ in Equation 6, there are essentially two cases to consider: either we stop decomposing the space $\mathcal{F}$ (corresponding to the $\{\mathcal{F}\}$ term in Equation 6) or we continue to split it further. This observation naturally suggests the use of a hierarchical prior, which recursively subdivides the remaining prior weight amongst each of the two possible choices. If we use a uniform weighting for each possibility, this gives a prior weighting of $2^{-\Gamma_d(f)}$, where $\Gamma_d(f)$ returns the total number of stop/split decisions needed to describe the factorization $f \in \mathcal{C}_d(\mathcal{F})$, with the base case of $\Gamma_0(f) := 0$ (since when $d = 0$, no stop or split decision needs to be made). This prior weighting is identical to how the Context Tree Weighting method (Willems et al., 1995) weights over tree structures, and is an application of the general technique used by the class of Tree Experts described in Section 5.3 of (Cesa-Bianchi & Lugosi, 2006). It is a valid prior, as one can show $\sum_{f \in \mathcal{C}_d} 2^{-\Gamma_d(f)} = 1$ for all $d \in \mathbb{N}$.

One appealing side-effect of this recursive construction is that it assigns more prior weight towards factorizations containing smaller amounts of nested substructure. For instance, returning again to our example, $2^{-\Gamma_2(\mathcal{O})} = \frac{1}{2}$, while $2^{-\Gamma_2(\mathcal{O}_1 \times \mathcal{O}_{2,1} \times \mathcal{O}_{2,2})} = \frac{1}{8}$. A second, less obvious property is that the prior implicitly contains some structure that will make model averaging easier. How this works in our reinforcement learning setting will become clear in the next section.

3.2. Recursively Factored Mixture Environment Models

Now that we have our prior over factorizations, we proceed by combining it with the model averaging tech-

nique described in Section 2.3 to define the class of recursively factored mixture environment models.

The first step is to commit to a set of base environment model factors from which each factored environment is formed. More precisely, this requires specifying an environment model factor for each element of the set of possible stopping points

$$\mathcal{S}_d(\mathcal{F}) := \{\mathcal{F}\} \cup \bigcup_{i=1}^{\dim(\mathcal{F})} \mathcal{S}_{d-1}(\mathcal{F}_i) \qquad \text{for } d > 0,$$

with $\mathcal{S}_0(\mathcal{F}) := \{\mathcal{F}\}$, within a recursively decomposable space $\mathcal{F}$ of nesting depth d. In our previous example, $\mathcal{S}_2(\mathcal{O}) = \{\mathcal{O}, \mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_{1,1}, \mathcal{O}_{1,2}, \mathcal{O}_{2,1}, \mathcal{O}_{2,2}\}$, which means that we would need to specify 7 environment model factors. Each environment model factor needs to have an appropriate type signature that depends on both the history and the parts of the percept space preceding it. For instance, an environment model factor processing $\mathcal{O}_2$ at time n in our previous example would depend on a history string that was an element of the set $(\mathcal{A} \times \mathcal{O})^{n-1} \times \mathcal{A} \times \mathcal{O}_1$. Similarly, an environment model factor for $\mathcal{O}_{2,2}$ would depend on a history string from the set $(\mathcal{A} \times \mathcal{O})^{n-1} \times \mathcal{A} \times \mathcal{O}_1 \times \mathcal{O}_{2,1}$. Now, given a history $ax_{1:n}$ and a $d \geq 0$ times recursively decomposable space $\mathcal{F}$, a recursively factored mixture environment is defined as

$$\xi^d_{\mathcal{F}}(x_{1:n} \,|\, a_{1:n}) := \sum_{f \in \mathcal{C}_d(\mathcal{F})} 2^{-\Gamma_d(f)} \rho_f(x_{1:n} \,|\, a_{1:n}), \quad (7)$$

where each factored model is defined by a product of environment model factors

$$\rho_f(x_{1:n} \,|\, a_{1:n}) := \prod_{\tau \in \mathcal{T}(f)} \pi_\tau(x^\tau_{1:n} \,|\, a_{1:n}).$$

Here $\mathcal{T}(\mathcal{X}) := \{\mathcal{X}_i\}_{i=1}^k$ denotes the set of subspaces of a given factored percept space $\mathcal{X}$ and $x^{\mathcal{X}_i}_{1:n} \in \mathcal{X}_i^n$ denotes the component of $x_{1:n}$ corresponding to $\mathcal{X}_i$. The base case of Equation 7, when $d = 0$, yields $\xi^0_{\mathcal{F}}(x_{1:n} \,|\, a_{1:n}) = \pi_{\mathcal{F}}(x_{1:n} \,|\, a_{1:n})$.

Notice that there exists a significant amount of shared structure in Equation 7, since each environment model factor can appear multiple times in the definition of each factored environment model. This property, along with the recursive definition of our prior in Section 3.1, allows us to derive the identity

$$2\xi^d_{\mathcal{F}}(x_{1:n} \,|\, a_{1:n}) = \pi_{\mathcal{F}}(x^{\mathcal{F}}_{1:n} \,|\, a_{1:n}) + \prod_{i=1}^{\dim(\mathcal{F})} \xi^{d-1}_{\mathcal{F}_i}(x_{1:n} \,|\, a_{1:n}). \quad (8)$$

This identity, whose proof we defer to the appendix, constitutes the core theoretical result of this work: it allows Equation 7 to be computed efficiently.

Equation 8 is essentially an application of the Generalized Distributive Law (Aji & McEliece, 2000), a key computational concept underlying many efficient algorithms. By using dynamic programming to compute each $\xi^d_{\mathcal{F}}$ term only once, the time overhead of performing exact model averaging over $\mathcal{C}_d(\mathcal{F})$ is reduced to just $O(n|\mathcal{S}_d(\mathcal{F})|)$. Furthermore, provided each base environment model factor can be updated online and the $\{\xi^d_{\mathcal{F}'}\}_{\mathcal{F}' \in \mathcal{S}_d(\mathcal{F})}$ terms are kept in memory, each percept can be processed online in time $O(|\mathcal{S}_d(\mathcal{F})|)$. A concrete application of this method will be given in Section 4.

Finally, as our technique performs exact model averaging, it is straightforward to provide theoretical performance guarantees along the lines of Section 2.3.

Theorem 1. *Given a recursively decomposable space $\mathcal{F}$ with nesting depth $d \in \mathbb{N}$, for all $a_{1:n} \in \mathcal{A}^n$, for all $x_{1:n} \in \mathcal{X}^n$, for all $f \in \mathcal{C}_d(\mathcal{F})$, we have*

$$-\log_2 \xi^d_{\mathcal{F}}(x_{1:n} \,|\, a_{1:n}) \leq \Gamma_d(f) \; - \log_2 \rho_f(x_{1:n} \,|\, a_{1:n}),$$

and for any environment μ,

$$D_{1:n}(\mu \,\|\, \xi^d_{\mathcal{F}}) \leq \Gamma_d(f) + D_{1:n}(\mu \,\|\, \rho_f).$$

Hence our technique is asymptotically competitive with the best factorization in $\mathcal{C}_d(\mathcal{F})$.

4. Quad-Tree Factorization

Section 3 provides a general framework for specifying recursively factored mixture environment models. The aim of this current section is to give an example of how this framework can extend existing model learning algorithms to domains with large observation spaces, such as Atari 2600 games. The quad-tree factorization (QTF) technique we now describe is particularly suited to image-based observation spaces. Although our factorization is presented in Atari 2600 terms, it is easily extended to other domains whose observation space exhibits two-dimensional structure.

Following the notation used by Bellemare et al. (2012a) to describe Atari 2600 observation spaces, let $\mathcal{D}_x$ and $\mathcal{D}_y$ denote totally ordered sets of row and column indices, with the joint index space given by $\mathcal{D} := \mathcal{D}_x \times \mathcal{D}_y$. In our case, $\mathcal{D}_x = \{1, 2, \ldots, 160\}$ and $\mathcal{D}_y = \{1, 2, \ldots, 210\}$. Denote by $\mathcal{C}$ a finite set of possible pixel colours. We define a pixel as a tuple $(x, y, c) \in \mathcal{D}_x \times \mathcal{D}_y \times \mathcal{C}$, where x and y denote the pixel's row and column location and c describes its colour. An observation o is defined as a set of $|\mathcal{D}|$ pixels, with each location $(x, y) \in \mathcal{D}$ uniquely corresponding to a single pixel (x, y, c). The set of all possible observations is denoted by $\mathcal{O}$.

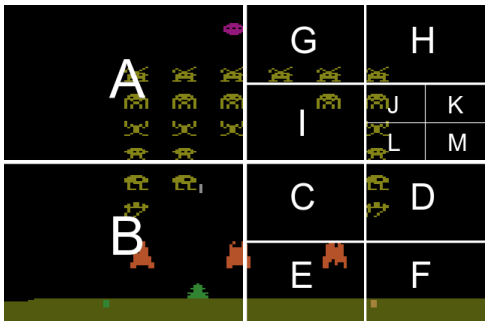


Figure 2. An example quad-tree factorization of an Atari 2600 screen, where each labelled square corresponds to a factor. The game shown is SPACE INVADERS.

We now describe a natural way to recursively decompose $\mathcal{O}$ by dividing each region into four equal parts; an example of such a decomposition is shown in Figure 2. For a given $\mathcal{D}'_x \subseteq \mathcal{D}_x$ whose ordered elements are $x_1, x_2, x_3, \ldots, x_n$, denote by $\mathrm{l}(\mathcal{D}'_x) := \{x_1, x_2, \ldots, x_{\lfloor n/2 \rfloor}\}$ and by $\mathrm{h}(\mathcal{D}'_x) := \{x_{\lfloor n/2 \rfloor + 1}, x_{\lfloor n/2 \rfloor + 2}, \ldots, x_n\}$ the lower and upper halves of $\mathcal{D}'_x$; similarly let $\mathrm{l}(\mathcal{D}'_y)$ and $\mathrm{h}(\mathcal{D}'_y)$ denote the two halves of $\mathcal{D}'_y \subseteq \mathcal{D}_y$. Let $\mathcal{O}_{\mathcal{D}'_x, \mathcal{D}'_y} := \{\{(x, y, c) : x \in \mathcal{D}'_x, y \in \mathcal{D}'_y, (x, y, c) \in o\} : o \in \mathcal{O}\}$ be the set of image patches that can occur in the region defined by $\mathcal{D}'_x$ and $\mathcal{D}'_y$, such that $\mathcal{O} = \mathcal{O}_{\mathcal{D}_x, \mathcal{D}_y}$. Assuming for now that $|\mathcal{D}_x|$ and $|\mathcal{D}_y|$ are both divisible by 2^d, the recursively decomposable space $\mathcal{F}^d_{\mathcal{D}_x, \mathcal{D}_y}$ on $\mathcal{O}$ is then recursively defined as

$$\mathcal{F}^d_{\mathcal{D}'_x, \mathcal{D}'_y} := \begin{cases} O_{\mathcal{D}'_x, \mathcal{D}'_y} & \text{if } d = 0 \\ \mathcal{F}^{d-1}_{\mathrm{l}(\mathcal{D}'_x), \mathrm{l}(\mathcal{D}'_y)} \times \mathcal{F}^{d-1}_{\mathrm{l}(\mathcal{D}'_x), \mathrm{h}(\mathcal{D}'_y)} \times & \\ \quad \mathcal{F}^{d-1}_{\mathrm{h}(\mathcal{D}'_x), \mathrm{l}(\mathcal{D}'_y)} \times \mathcal{F}^{d-1}_{\mathrm{h}(\mathcal{D}'_x), \mathrm{h}(\mathcal{D}'_y)} & \text{otherwise} \end{cases} \tag{9}$$

The base case corresponds to indivisible image patches of size $|\mathcal{D}_x|/2^d \times |\mathcal{D}_y|/2^d$. These image patches are used to form larger image patches. Note that there is a one-to-one correspondence between elements of $\mathcal{F}^d_{\mathcal{D}_x, \mathcal{D}_y}$ and $\mathcal{O}$. Whenever $|\mathcal{D}_x|$ or $|\mathcal{D}_y|$ is not divisible by 2^d, as is the case with Atari 2600 games, a simple solution is to enlarge $\mathcal{D}_x$ and $\mathcal{D}_y$ appropriately and insert pixels whose colour is a special *out-of-screen* indicator[1].

4.1. Algorithm

Algorithm 1 provides pseudocode for an online implementation of QTF. The algorithm is invoked once per time step; its arguments are the top-level space $\mathcal{F}^d_{\mathcal{D}_x, \mathcal{D}_y}$, its nesting depth d, and the current percept. The algorithm recursively updates the base environment model factor $\pi_{\mathcal{F}}$ corresponding to $\mathcal{F}$ as well as its factors $\{\mathcal{F}_i\}_{i=1}^4$. The $\pi_{\mathcal{F}}$ and $\xi^d_{\mathcal{F}}$ variables respectively store the values $\pi_{\mathcal{F}}(x^{\mathcal{F}}_{1:n} \,|\, a_{1:n})$ and $\xi^d_{\mathcal{F}}(x_{1:n} \,|\, a_{1:n})$; both sets of variables are initialized to 1. As QTF is a meta-algorithm, any suitable set of base environment model factors may be used. In our implementation we avoid numerical issues such as underflow by storing and manipulating probabilities in the logarithmic domain.

Algorithm 1 Online Quad-Tree Factorization

Require: A quad-tree decomposable space $\mathcal{F}$
Require: A nesting depth $d \in \{0\} \cup \mathbb{N}$
Require: A percept x_t at time $t \in \mathbb{N}$

QTF$(\mathcal{F}, d, x_t)$
  Update $\pi_{\mathcal{F}}$ with $x^{\mathcal{F}}_t$
  if $d = 0$ **then**
    $\xi^d_{\mathcal{F}} \leftarrow \pi_{\mathcal{F}}$
  else
    for $i = 1 \ldots 4$ **do**
      QTF$(\mathcal{F}_i, d-1, x_t)$
    $\xi^d_{\mathcal{F}} \leftarrow \frac{1}{2}\pi_{\mathcal{F}} + \frac{1}{2}\prod_{i=1}^{4} \xi^{d-1}_{\mathcal{F}_i}$

5. Experiments

We evaluated our quad-tree factorization on the Arcade Learning Environment (ALE) (Bellemare et al., 2013), a reinforcement learning interface to more than sixty Atari 2600 games. The observation space we consider is the 160×210-pixel game screen, with each pixel taking on one of 128 colours. When emulating in real-time, game screens are generated at the rate of 60 frames per second; a full five-minute episode lasts 18,000 frames. The action space is the set of joystick motions and button presses, giving a total of 18 distinct actions. Similar to previous work using ALE (Bellemare et al., 2013; Naddaf, 2010; Bellemare et al., 2012a;b), we designed and optimized our algorithm using five training games and subsequently evaluated it on a hold-out set of fifteen testing games.

5.1. Experimental Setup

For practical reasons, we used a quad-tree factorization that considered image patches whose size ranged from 32×32 down to 4×4; this corresponds to a nesting depth of 3. Empirically, we found that larger patch sizes generalized poorly and smaller patches performed

[1] Having each dimension divisible by 2^d is not strictly required, but we found it to considerably simplify the implementation details.

worse due to limited contextual information. Each patch was predicted using a context tree switching (CTS) model (Veness et al., 2012) based on ten neighbouring patches from the current and previous time steps, similar to the $\mathcal{P}$-context trees of Veness et al. (2011). We also encoded the most recent action as an input feature to the CTS model. To handle the large image patch alphabets, we used the recently developed Sparse Sequential Dirichlet (SSD) estimator (Veness & Hutter, 2012) at the internal nodes of our CTS model.

To curb memory usage and improve sample efficiency, we incorporated a set of well-established techniques into our implementation. For each patch size, a single CTS model was shared across patch locations, giving a limited form of translation invariance. Finally, each CTS model was implemented using hashing (Willems & Tjalkens, 1997) so as to better control memory usage. In practice, we found that larger hash tables always resulted in better results.

We compared QTF with factored models that used a fixed patch size (4×4, 8×8, 16×16, 32×32). Each model was trained on each game for 10 million frames, using a policy that selected actions uniformly at random and then executed them for k frames, where k was also chosen uniformly at random from the set $K := \{4, 8, 12, 16\}$. This policy was designed to visit more interesting parts of the state space and thus generate more complex trajectories. The models received a new game screen every fourth frame, yielding 15 time steps per second of play. Predicting at higher frame rates was found to be easier, but led to qualitatively similar results while requiring more wall-clock time per experiment.

5.2. Results

After training, we evaluated the models' ability to predict the future conditional on action sequences. This evaluation phase took place over an additional 8000 frames, with action sequences again drawn by selecting actions uniformly at random and executing them for $k \in K$ frames. At each time step t, each model was asked to predict 90 steps ahead and recorded how many frames were perfectly generated. Predictions at $t+k$ were thus made using the true history up to time t and the $k-1$ already sampled frames.

Table 1 summarizes the result of this evaluation. Which patch size best predicts game screens depends on a number of factors, such as the size of in-game objects, their frame to frame velocity and the presence of animated backgrounds. QTF achieves a level of performance reasonably close to the best patch size, above 95% in 13 out of 20 games. In some games, QTF even improves on the performance of the best fixed-size model. Ultimately, Theorem 1 ensures that QTF will asymptotically achieve a prediction accuracy comparable to the best decomposition available to it.

Table 1. Average number of forward time steps correctly predicted for the fixed-size and QTF models. The first five games constitute our training set. Highest per-game accuracy is indicated in bold blue.

	Time Steps Correct				
	4×4	8×8	16×16	32×32	QTF
Asterix	0.748	1.17	1.11	**1.44**	**1.44**
Beam Rider	≈ 0	0.007	0.100	**0.227**	0.185
Freeway	0.303	**4.64**	2.21	1.63	3.20
Seaquest	0.019	0.427	0.157	**2.39**	1.85
Space Invaders	0.606	0.502	0.736	0.274	**0.830**
Amidar	13.0	12.9	13.1	13.3	**13.4**
Crazy Climber	0.347	0.758	0.650	2.50	**2.66**
Demon Attack	0.004	0.004	0.006	0.013	**0.026**
Gopher	0.048	0.375	0.747	**2.27**	**2.28**
Krull	0.014	0.083	0.482	**1.14**	0.233
Kung-Fu Master	2.87	3.17	3.35	**3.57**	3.53
Ms. Pacman	0.047	0.096	0.264	0.434	**0.463**
Pong	0.319	1.79	2.50	**3.99**	3.97
Private Eye	0.000	≈ 0	0.001	**0.010**	0.003
River Raid	0.513	0.643	0.672	**1.76**	1.68
Star Gunner	0.041	0.417	0.659	**1.48**	0.802
Tennis	3.04	4.35	4.01	**4.94**	4.74
Up and Down	0.354	0.704	1.77	1.64	**2.20**
Wizard of Wor	0.957	0.946	**0.961**	0.520	0.519
Yars' Revenge	≈ 0	0.003	0.005	0.005	**0.008**

Sequential, probabilistic prediction algorithms such as QTF are often evaluated based on their n-step average logarithmic loss, defined as $\frac{1}{n}\sum_{i=1}^{n} -\log_2 \xi(x_i \,|\, ax_{<i}a_i)$ for an environment model ξ. This measure has a natural information theoretic interpretation: on average, losslessly encoding each percept requires this many bits. While counting the number of correct future frames is a conservative measure and is heavily influenced by the sampling process, the logarithmic loss offers a more fine-grained notion of predictive accuracy. As shown in Table 2, our quad-tree factorization achieves significantly lower per-frame loss than any fixed-size model. In view that each frame contains $160 \times 210 \times 7 = 235,200$ bits of data, our gains in Pong and Freeway – about 3 bits per frame – are particularly important.

6. Discussion

The benefits of predicting observations using environment model factors corresponding to larger image patches are twofold. Large regular patterns, such as the invaders in Space Invaders, can easily be represented as a single symbol. When using a base model

Table 2. Per-frame logarithmic loss for the fixed-size and QTF models, averaged over the whole training sequence. The first five games constitute our training set. The lowest loss is indicated in bold blue.

	Logarithmic Loss (Base 2)				
	4×4	8×8	16×16	32×32	QTF
Asterix	81.83	29.44	188.2	2166	**17.77**
Beam Rider	850.6	335.4	710.8	4059	**68.63**
Freeway	12.68	8.19	50.88	251.1	**3.05**
Seaquest	101.8	173.9	1328	7887	**51.26**
Space Invaders	161.3	129.5	678.2	8698	**27.99**
Amidar	35.74	36.58	168.1	815.9	**9.95**
Crazy Climber	205.5	161.0	446.9	2172	**40.84**
Demon Attack	716.9	1587	5502	19680	**531.1**
Gopher	72.85	28.24	66.02	608.9	**9.97**
Krull	615.4	1103	4025	15220	**245.2**
Kung-Fu Master	74.62	59.33	179.9	1012	**20.87**
Ms. Pacman	109.2	183.5	1053	6362	**48.9**
Pong	33.75	13.71	23.07	121.0	**3.24**
Private Eye	453.5	623.4	1922	7956	**162.6**
River Raid	298.5	256.5	1034	6055	**77.35**
Star Gunner	438.0	481.2	1980	10790	**139.2**
Tennis	178.5	290.7	945.6	4134	**93.6**
Up and Down	1461	2220	5104	14490	**854.7**
Wizard of Wor	134.3	81.98	277.9	1778	**26.42**
Yars' Revenge	667.4	1251	3264	20810	**493.3**

such as CTS, which treats symbols atomically, *sampling* from QTF is often faster as fewer symbols need to be generated. Thus our quad-tree factorization produces a forward model of Atari 2600 games which is both efficient and accurate.

In our experiments, we used Context Tree Switching (CTS) models as our environment model factors. One limitation of this approach is that CTS has no provision for partial symbol matches: altering a single pixel within an image patch yields a completely new symbol. This presents a significant difficulty, as the size of the alphabet corresponding to $\mathcal{F}$ grows exponentially with its nesting depth d. The results of Table 2 are symptomatic of this issue: larger patch size models tend to suffer higher loss. As our meta-algorithm is independent of the choice of environment model factors, other base models perhaps better suited to noisy inputs may improve predictive accuracy, for example locally linear models (Farahmand et al., 2009), dynamic bayes networks (Walsh et al., 2010), and neural network architectures (Sutskever et al., 2008).

7. Conclusion

In this paper, we introduced the class of recursively decomposable factorizations and described an algorithm that performs efficient Bayesian model averaging in a reinforcement learning setting. We instantiated our technique into a recursive quad-tree factorization of two-dimensional image spaces which we used to learn forward models of 160×210-pixel Atari 2600 games. To the best of our knowledge, our QTF model is the first to tackle observation spaces of this magnitude within the model-based reinforcement learning setting. We also expect that many existing algorithms can be improved by an appropriate combination with QTF.

While our results show that QTF allows us to obviate choosing a factorization a priori, there remains the question of how to best act given such a model of the environment. Recently, Joseph et al. (2013) argued that, when no member of the model class can represent the true environment, a mismatch arises between model accuracy and model usefulness: the best policy may not rely on the most accurate model. In the future, we will be integrating our modeling techniques with simulation-based search to further explore this issue in Atari 2600 games.

A. Appendix

A.1. Proof of Equation 8

Proof. Along the lines of the argument used to prove Lemma 2 in (Willems et al., 1995), we can rewrite

$$
\begin{aligned}
\xi^d_{\mathcal{F}}(x_{1:n} \mid a_{1:n}) &= \sum_{f \in \mathcal{C}_d(\mathcal{F})} 2^{-\Gamma_d(f)} \rho_f(x_{1:n} \mid a_{1:n}) \\
&= \sum_{f \in \mathcal{C}_d(\mathcal{F})} 2^{-\Gamma_d(f)} \prod_{\tau \in \mathcal{T}(f)} \pi_\tau(x^\tau_{1:n} \mid a_{1:n}) \\
&= \frac{1}{2}\pi_{\mathcal{F}}(x^{\mathcal{F}}_{1:n} \mid a_{1:n}) + \sum_{f \in \mathcal{C}_d \setminus \{\mathcal{F}\}} 2^{-\Gamma_d(f)} \prod_{\tau \in \mathcal{T}(f)} \pi_\tau(x^\tau_{1:n} \mid a_{1:n}) \\
&= \frac{1}{2}\pi_{\mathcal{F}}(x^{\mathcal{F}}_{1:n} \mid a_{1:n}) + \\
&\quad \frac{1}{2} \sum_{\substack{f_1 \in \\ \mathcal{C}_{d-1}(\mathcal{F}_1)}} \cdots \sum_{\substack{f_{\dim(\mathcal{F})} \in \\ \mathcal{C}_{d-1}(\mathcal{F}_{\dim(\mathcal{F})})}} \prod_{i=1}^{\dim(\mathcal{F})} 2^{-\Gamma_{d-1}(f_i)} \prod_{\tau \in \mathcal{T}(f_i)} \pi_\tau(x^\tau_{1:n} \mid a_{1:n}) \\
&= \frac{1}{2}\pi_{\mathcal{F}}(x^{\mathcal{F}}_{1:n} \mid a_{1:n}) + \\
&\quad \frac{1}{2} \prod_{i=1}^{\dim(\mathcal{F})} \left(\sum_{f \in \mathcal{C}_{d-1}(\mathcal{F}_i)} 2^{-\Gamma_{d-1}(f)} \rho_f(x_{1:n} \mid a_{1:n}) \right) \\
&= \frac{1}{2}\pi_{\mathcal{F}}(x^{\mathcal{F}}_{1:n} \mid a_{1:n}) + \frac{1}{2} \prod_{i=1}^{\dim(\mathcal{F})} \xi^{d-1}_{\mathcal{F}_i}(x_{1:n} \mid a_{1:n}).
\end{aligned}
$$

References

Aji, Srinivas M. and McEliece, Robert J. The Generalized Distributive Law. *IEEE Transactions on Information*

Theory, 46(2):325–343, 2000.

Asmuth, John and Littman, Michael L. Learning is Planning: Near Bayes-optimal Reinforcement Learning via Monte-Carlo Tree Search. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 2011.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. Investigating Contingency Awareness Using Atari 2600 Games. In *Proc. of the 26th AAAI Conference on Artificial Intelligence*, 2012a.

Bellemare, Marc G., Veness, Joel, and Bowling, Michael. Sketch-Based Linear Value Function Approximation. In *Advances in Neural Information Processing Systems 25*, 2012b.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research (JAIR), to appear*, 2013.

Cesa-Bianchi, Nicolo and Lugosi, Gabor. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006.

Diuk, Carlos, Li, Lihong, , and Leffler, R. Bethany. The Adaptive k-Meteorologists Problem and Its Application to Structure Learning and Feature Selection in Reinforcement Learning. In *Proc. of the 26th International Conference on Machine learning*, 2009.

Doshi-Velez, Finale. The Infinite Partially Observable Markov Decision Process. In *Advances in Neural Information Processing Systems 22*, 2009.

Farahmand, Amir massoud, Shademan, Azad, Jägersand, Martin, and Csaba Szepesvári. Model-based and Model-free Reinforcement Learning for Visual Servoing. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2009.

Grünwald, Peter D. *The Minimum Description Length Principle (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

Guez, Arthur, Silver, David, and Dayan, Peter. Efficient Bayes-Adaptive Reinforcement Learning using Sample-based Search. In *Advances in Neural Information Processing Systems 25*, 2012.

Hausknecht, Matthew, Khandelwal, Piyush, Miikkulainen, Risto, and Stone, Peter. HyperNEAT-GGP: A HyperNEAT-based Atari general game player. In *Proc. of the Genetic and Evolutionary Computation Conference*, 2012.

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

Joseph, Joshua, Geramifard, Alborz, Roberst, John W., How, Jonathan P., and Roy, Nicholas. Reinforcement Learning with Misspecified Model Classes. In *Proc. of the IEEE International Conference on Robotics and Automation*, 2013.

Kaelbling, Leslie Pack, Littman, Michael L., and Cassandra, Anthony R. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101: 99–134, 1998.

Kocsis, Levente and Szepesvári, Csaba. Bandit Based Monte-Carlo Planning. In *Proc. of the European Conference on Machine Learning*, 2006.

Naddaf, Yavar. Game-Independent AI Agents for Playing Atari 2600 Console Games. Master's thesis, University of Alberta, 2010.

Nguyen, Phuong, Sunehag, Peter, and Hutter, Marcus. Context Tree Maximizing Reinforcement Learning. In *Proc. of the 26th AAAI Conference on Artificial Intelligence*, pp. 1075–1082, Toronto, 2012. AAAI Press. ISBN 978-1-57735-568-7.

Poupart, Pascal. Model-based Bayesian Reinforcement Learning in Partially Observable Domains. In *Proc. of the International Symposium on Artificial Intelligence and Mathematics*, 2008.

Puterman, Martin L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Ross, Stéphane and Pineau, Joelle. Model-Based Bayesian Reinforcement Learning in Large Structured Domains. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 2008.

Silver, David and Veness, Joel. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems 23*, 2010.

Silver, David, Sutton, Richard S., and Müller, Martin. Sample-based Learning and Search with Permanent and Transient Memories. In *Proc. of the 25th International Conference on Machine Learning*, 2008.

Silver, David, Sutton, Richard S., and Müller, Martin. Temporal-Difference Search in Computer Go. *Machine Learning*, 87(2):183–219, 2012.

Sutskever, Ilya, Hinton, Geoffrey, and Taylor, Graham. The Recurrent Temporal Restricted Boltzmann Machine. In *Advances in Neural Information Processing Systems 21*, 2008.

Sutton, Richard S. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bulletin*, 2 (4):160–163, 1991.

Veness, Joel and Hutter, Marcus. Sparse Sequential Dirichlet Coding. *ArXiv e-prints*, 2012.

Veness, Joel, Ng, Kee Siong, Hutter, Marcus, and Silver, David. Reinforcement Learning via AIXI Approximation. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2010.

Veness, Joel, Ng, Kee Siong, Hutter, Marcus, Uther, William, and Silver, David. A Monte-Carlo AIXI Approximation. *Journal of Artificial Intelligence Research (JAIR)*, 40(1), 2011.

Veness, Joel, Ng, Kee Siong, Hutter, Marcus, and Bowling, Michael. Context Tree Switching. In *Proc. of the Data Compression Conference (DCC)*, 2012.

Walsh, Thomas J., Goschin, Sergiu, and Littman, Michael L. Integrating Sample-based Planning and Model-Based Reinforcement Learning. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2010.

Willems, F. and Tjalkens, T.J. Complexity Reduction of the Context-Tree Weighting Algorithm: A Study for KPN Research. *EIDMA Report RS.97.01*, 1997.

Willems, Frans M.J., Shtarkov, Yuri M., and Tjalkens, Tjalling J. The Context Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory*, 41:653–664, 1995.