In *The Sixth International Conference on Autonomous Agents and Multiagent Systems* (AAMAS 07), Honolulu, Hawai'i, May 2007.

Model-Based Function Approximation in Reinforcement Learning

Nicholas K. Jong
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-0233
nkj@cs.utexas.edu

Peter Stone
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-0233
pstone@cs.utexas.edu

ABSTRACT

Reinforcement learning promises a generic method for adapting agents to arbitrary tasks in arbitrary stochastic environments, but applying it to new real-world problems remains difficult, a few impressive success stories notwithstanding. Most interesting agent-environment systems have large state spaces, so performance depends crucially on efficient generalization from a small amount of experience. Current algorithms rely on model-free function approximation, which estimates the long-term values of states and actions directly from data and assumes that actions have similar values in similar states. This paper proposes model-based function approximation, which combines two forms of generalization by assuming that in addition to having similar values in similar states, actions also have similar effects. For one family of generalization schemes known as averagers, computation of an approximate value function from an approximate model is shown to be equivalent to the computation of the exact value function for a finite model derived from data. This derivation both integrates two independent sources of generalization and permits the extension of model-based techniques developed for finite problems. Preliminary experiments with a novel algorithm, AMBI (Approximate Models Based on Instances), demonstrate that this approach yields faster learning on some standard benchmark problems than many contemporary algorithms.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*dynamic programming*

General Terms

Algorithms, Experimentation

Keywords

function approximation, models, reinforcement learning

1. INTRODUCTION

Research into reinforcement learning (RL) [16] addresses an extraordinarily general problem: how to learn good agent behaviors in unknown stochastic environments. A typical RL algorithm adapts to an arbitrary agent-environment system by learning the optimal value function, which estimates the cumulative reward possible from each system state for each initial action. After learning, an agent can behave optimally simply by always choosing an action that maximizes this function at the current state. In practical applications, the learning algorithm must rely on a relatively small amount of data to compute an accurate value function defined over a large (often infinite) state space. *Generalization* thus plays a critical role in determining the effectiveness of RL in the real world.

A large body of RL research studies the interaction between RL algorithms and *function approximation*, which permits the finite representation of value functions defined over infinite domains. Current RL function approximators rely on the inductive bias that actions have similar values in similar states. This paper proposes a *model-based* approach that utilizes the additional inductive bias that actions have similar effects in similar states. This approach is motivated by the observation that in many practical applications, the dynamics of the system affords more opportunities for generalization than the optimal value function. For example, in robot navigation, a turn action may have the same immediate effect on the agent's pose regardless of its state, but its value depends on the action's pose, environment, and goal.

Model-based reasoning has previously been applied to RL in finite systems but the absence of efficient representations for learned models of infinite stochastic systems has hampered extensions to this setting. This paper shows how to avoid reasoning with unwieldy infinite models by composing certain forms of model approximation and function approximation, using a strategy with three conceptual steps:

1. Define an approximate (infinite) model of the original system using experience data.
2. Define an approximate value function using a finite model derived from the infinite model and a finite sample of states.
3. Compute the optimal value function of the finite model.

This paper contributes techniques for accomplishing Step 1 of the above strategy. By leveraging existing work for Steps 2 and 3, it also contributes a novel algorithm, AMBI, which implements the strategy to learn online in infinite systems. Section 2 introduces notation and prior work. Section 3 formally presents model-based function approximation. Section 4 describes the AMBI algorithm. Section 5 evaluates the performance of this algorithm on some benchmark systems. Section 6 situates the contribution of this paper in the literature, and Section 7 concludes.

2. BACKGROUND

This section describes the prior work upon which model-based function approximation builds. It first describes the computation of value functions for known finite systems (Step 3 of the strategy outlined in Section 1), using the formalism of Markov decision problems. It then describes a technique, fitted value iteration, for computing value functions in known infinite systems (Step 2).

2.1 Markov Decision Problems

Classical RL algorithms assume that the agent-environment system constitutes a Markov decision process (MDP) [13]. An MDP $M = \langle S, A, \mathcal{P}, \mathcal{R}\rangle$ comprises a finite set of states S, a finite set of actions A, a family of transition probability functions $\mathcal{P}$, and a family of reward functions $\mathcal{R}$. The effect of executing an action $a \in A$ depends only on the current state $s \in S$. The reward function $\mathcal{R}^a : S \to \mathbb{R}$ specifies the average immediate reward $\mathcal{R}^a(s) = \mathcal{R}^a_s$ for executing a in s. The transition function $\mathcal{P}^a : S \to \Delta(S)$ specifies the probability mass function $\mathcal{P}^a(s) = \mathcal{P}^a_s$ over successor states, so action a transitions s to $s' \in S$ with probability $\mathcal{P}^a_s(s')$.

A policy $\pi : S \to A$ describes the behavior of an agent and induces a value function $V^\pi : S \to \mathbb{R}$ that maps each state $s \in S$ to the expected cumulative reward $V^\pi(s) = \mathcal{R}^{\pi(s)}_s + \gamma \sum_{s' \in S} \mathcal{P}^{\pi(s)}_s(s')V^\pi(s')$, where $\gamma \in [0, 1]$ is a discount factor sometimes necessary to ensure the existence of the value function. The action value function $Q^\pi : S \times A \to \mathbb{R}$ maps each state-action pair to the expected cumulative reward due to executing a in s and then behaving according to π, $Q^\pi(s, a) = \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_s(s')V^\pi(s')$. The optimal value function V^* satisfies the Bellman optimality equations $V^*(s) = \max_a Q^*(s, a)$ for all $s \in S$ and allows an agent easily to obtain an optimal policy π^* by choosing $\pi^*(s) \in \text{argmax}_a Q^*(s, a)$.

An offline algorithm known as value iteration can compute V^* relatively efficiently for an arbitrary finite system, given the MDP [9]. It computes a sequence of value functions $\hat{V}_i$ that converge to V^*, according to the following recurrence relations: for all $s \in S$ and $a \in A$, $\hat{V}_0(s) = 0$, $\hat{Q}_i(s, a) = \mathcal{R}^a_s + \gamma \sum_{s' \in S} \mathcal{P}^a_s(s')\hat{V}_i(s')$, and $\hat{V}_{i+1}(s) = \max_a \hat{Q}_i(s, a)$. Although this algorithm takes a known MDP as input, it serves as the basis for many RL algorithms, which take as input on the state space S and action space A. Model-free RL algorithms, such as Q-learning, converge to Q^* directly from experience data, which it uses in a stochastic approximation of the update rule of value iteration. Model-based RL algorithms, such as Prioritized Sweeping, incrementally estimate $\mathcal{P}$ and $\mathcal{R}$ given experience data and then perform an incremental form of value iteration on this learned model [11]. Reasoning about model uncertainty provided the first algorithms with finite-time PAC convergence guarantees [3, 7].

Extending the MDP formalism to continuous state spaces simply requires first, the redefinition of each $\mathcal{P}^a_s$ as a probability density function instead of a mass function, and second, the substitution of integration for summation in the definitions of the value functions V and Q. Model-free algorithms extend naturally to this setting, since they do not compute the summation or integration explicitly. Replacing the representation of the value function with a function approximator removes the convergence guarantees of algorithms such as Q-learning, but this approach has led to important RL success stories [18]. In contrast, model-based RL algorithms rely on offline algorithms, such as value iteration, that do not extend to continuous state spaces due to the impossibility of explicitly evaluating the value function at every state.

2.2 Fitted Value Iteration

One approach to adapting value iteration to continuous state spaces is known as fitted value iteration. This algorithm only explicitly estimates the value function at a finite sample of states $X \subset S$. It can then modify value iteration as given in Section 2.1 as follows. $\hat{V}_0$ is initialized arbitrarily. To obtain $\hat{V}_{i+1}$, an arbitrary function approximation generalizes training data of the form $\langle x, \max_a \hat{Q}_i(x, a)\rangle$ for all $x \in X$. Since $\hat{Q}_i(s, a) = \mathcal{R}^a_s + \gamma \int \mathcal{P}^a_s(s')\hat{V}_i(s')\,\mathrm{d}s'$ still requires integrating over successor states, fitted value iteration is most easily applied to deterministic systems, or other systems where the number of possible successor states is finite. Even in such cases, popular function approximators such as neural networks can cause fitted value iteration to diverge [2].

Gordon proved convergence for one class of function approximators called averagers [6]. An averager $\phi^X : S \to \Delta(X)$ maps each state s in the infinite state space to a probability mass function ϕ^X_s over sample states. It approximates $\hat{V}(s) = \sum_{x \in X} \phi^X_s(x)\hat{V}(x)$, where the parameters of the approximation scheme are the values $\hat{V}(x)$ of the sample X. For example, a k-nearest-neighbors approximation scheme would define ϕ^X_s as a uniform distribution over the k elements of X closest to s. The equation for the action value function then becomes

$$
\begin{aligned}
&\hat{Q}_i(s, a) \\
&\quad= \mathcal{R}^a_s + \gamma \int \mathcal{P}^a_s(s') \left(\sum_{x' \in X} \phi^X_{s'}(x')\hat{V}_i(x') \right) \mathrm{d}s' \\
&\quad= \mathcal{R}^a_s + \gamma \sum_{x' \in X} \left(\int \mathcal{P}^a_s(s')\phi^X_{s'}(x')\,\mathrm{d}s' \right) \hat{V}_i(x'). \qquad (1)
\end{aligned}
$$

Equation 1 is equivalent to the equation for the exact action value function for a derived finite MDP $\tilde{M} = \langle X, A, \tilde{\mathcal{P}}, \mathcal{R}\rangle$, where $\tilde{\mathcal{P}}^a_x(x') = \int \mathcal{P}^a_x(s')\phi^X_{s'}(x')\,\mathrm{d}s'$. Fitted value iteration with an averaging approximation scheme thus converges, if $\gamma < 1$ or if $\tilde{M}$ is well-behaved. Essentially, it first uses the approximation scheme ϕ^X and the original MDP M to construct a finite model $\tilde{M}$ (Figure 1), then computes the exact value function $\tilde{V} : X \to \mathbb{R}$ of $\tilde{M}$, and finally generalizes $\tilde{V}$ to an action value function $\hat{Q} : S \times A \to \mathbb{R}$ for the original MDP by substituting $\tilde{V}$ for $\hat{V}_i$ in Equation 1.

3. MODEL-BASED APPROXIMATION

Fitted value iteration with averagers converges despite the application of function approximation, but it still requires knowledge of the original MDP to construct the finite model over the sampled states. Thus, it does not apply directly to the RL problem. Nevertheless, some recently successful

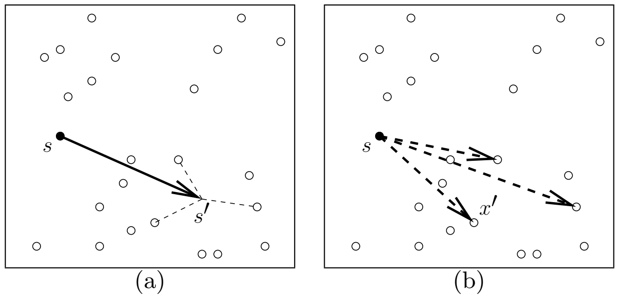


Figure 1: (a) An MDP M specifies a deterministic transition from state s to state s', whose value an averager approximates as the weighted average of nearby sample states. (b) The derived MDP $\tilde{M}$ specifies a stochastic transition from the same state to the sample states, with probabilities equal to their weights in the averager's approximation of s'.

model-free RL algorithms iterate over a given sample of data to achieve a degree of stability while using function approximation [8, 14]. The fixed sample contains implicit knowledge of the MDP that generated the data. This section describes an approach that explicitly defines an approximate model from a sample, permitting the application of fitted value iteration to RL (Step 1 of the strategy outlined in Section 1).

The goal is to approximate each transition probability distribution $\mathcal{P}_s^a$ and expected reward $\mathcal{R}_s^a$ using a sample of experience data D. Each instance $d \in D$ represents a unit of experience $d = \langle s_d, a_d, r_d, s'_d \rangle$ consisting of a state s_d the agent visited, the action a_d that it executed, the reward r_d it received, and the transition to state s'_d it observed.

In the absence of generalization, the estimated effect of a given action a at a given state s depends only on the subset $D_s^a = \{d \in D \mid s_d = s \wedge a_d = a\}$ of instances with the same state and action. Consider drawing an instance $d \in D_s^a$ uniformly at random. Then the maximum likelihood estimate $\hat{\mathcal{R}}_s^a$ of the expected immediate reward is the mean of the random variable r_d, $\hat{\mathcal{R}}_s^a = \mathrm{E}[r_d] = \frac{1}{|D_s^a|}\sum_{d \in D_s^a} r_d$. Similarly, the maximum-likelihood transition probability function assigns to successor state s' the probability $\frac{1}{|D_s^a|}$ for each instance d such that $s'_d = s'$, so $\hat{\mathcal{P}}_s^a(s') = \sum_{d \in D_s^a \mid s'_d = s'} \frac{1}{|D_s^a|} = \frac{|D_{ss'}^a|}{|D_s^a|}$, where $D_{ss'}^a = \{d \in D_s^a \mid s'_d = s'\}$. Substituting this discrete approximation for $\mathcal{P}_s^a$ yields a Monte Carlo approximation of the integral in Equation 1.

The primary difficulty with simple maximum-likelihood estimation is that most states will never be visited, and the visited states will typically only have data for one action. The estimates $\mathcal{R}_s^a$ and $\mathcal{P}_s^a$ must instead generalize from $D^a = \{d \in D \mid a_d = a\}$, the set of instances at which a was executed. Generalizing from experience at a sample of states to an approximate model at an arbitrary state is analogous to value-function approximation generalizing from the values of a sample of states to the value of an arbitrary state.

Averagers can be applied equally well to the problem of generalizing a value function and generalizing a model. Let $S^a = \{s_d \mid d \in D^a\}$ be the set of states at which action a has been applied. Suppose the existence of an averager $\phi^{S^a} : S \to \Delta(S^a)$ that maps each $s \in S$ to a probability distribution over these states S^a, weighting the similarity of states in S^a to s (Figure 2a). The approximate estimate for the expected immediate reward then becomes

$$\hat{\mathcal{R}}_s^a = \sum_{d \in D^a} \phi_s^{S^a}(s_d) \cdot r_d. \tag{2}$$

The approximation of the transition probabilities may be defined similarly:

$$\hat{\mathcal{P}}_s^a(s') = \sum_{d \in D^a \mid s'_d = s'} \phi_s^{S^a}(s_d). \tag{3}$$

Note that Equation 3 includes a summation only to properly treat the case when two instances in D^a transition to the same successor state s'. If the same state is never reached twice in the data, then this equation may be expressed simply as $\hat{\mathcal{P}}_s^a(s'_d) = \phi_s^{S^a}(s_d)$ for s'_d associated with some instance d and $\hat{\mathcal{P}}_s^a(s') = 0$ otherwise.

Preliminary experiments showed that in some cases the approximate model defined above generalizes poorly over relatively large distances. Equation 3 assumes that if action a has the same effect in states s and s_d, then a will transition s to s'_d (Figure 2b). This assumption holds for "absolute" actions that transition an entire set of similar system states to the same successor state. In practice, most AI systems have "relative" actions that apply some local modification to each state in a set of similar states. For example, STRIPS operators used in planning add or delete propositions to any problem state that satisfies their preconditions. The action models used in mobile robotics generalize across any pose in free space but specify effects in terms of incremental changes to position or heading. The following definition generalizes the change in state resulting from an action instead of generalizing the identity of the state resulting from an action, assuming a vector state space (Figure 2c):

$$\hat{\mathcal{P}}_s^a(s') = \sum_{d \in D^a \mid s + s'_d - s_d = s'} \phi_s^{S^a}(s_d). \tag{4}$$

As above, this equation may alternatively be expressed as $\hat{\mathcal{P}}_s^a(s') = \phi_s^{S^a}(s_d)$ if there exists exactly one $d \in D^a$ such that $s + s'_d - s_d = s'$.

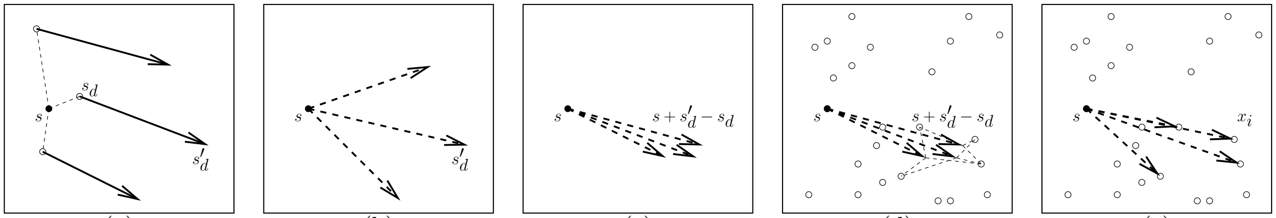
Equation 4 defines an approximate transition model in terms of sample data and an averager. Substituting $\hat{\mathcal{P}}_s^a$ in Equation 1 yields the equation for the exact action value function for a derived MDP $\tilde{M} = \langle X, A, \tilde{\mathcal{P}}, \tilde{\mathcal{R}} \rangle$ with the transition functions (Figures 2d and 2e) given by

$$\tilde{\mathcal{P}}_x^a(x') = \sum_{s'} \left(\sum_{d \in D^a \mid x + s'_d - s_d = s'} \phi_x^{S^a}(s_d) \right) \phi_{s'}^X(x') \tag{5}$$

and reward functions given by $\tilde{\mathcal{R}} = \hat{\mathcal{R}}$ (Equation 2). Note that the outer summation in Equation 5 formally sums over infinitely many $s' \in S$, but at most $|D^a|$ of the terms are nonzero for each x and a. These equations outline an RL algorithm that uses its experience D and averagers ϕ^{S^a} and ϕ^X to derive a model $\tilde{M}$ over a finite state set X, computes the optimal value function $\tilde{V}$ for $\tilde{M}$, and then generalizes $\tilde{V}$ to an approximate value function $\hat{Q}$ for the original system.

4. THE AMBI ALGORITHM

This section describes Approximate Models Based on Instances (AMBI), a RL algorithm that instantiates the outline from the previous section and the strategy in Section 1. Given any state $s \in S$, AMBI always chooses an action



(a) (b) (c) (d) (e)

Figure 2: (a) The averager ϕ^{S^a} approximates the effect of some action a at state s using nearby sample transitions $d \in D^a$. (b) The absolute-transition model predicts a stochastic transition to one of the s'_d. (c) The relative-transition model, which this paper adopts, predicts a stochastic transition with the same change in state as one of the sample transitions. (d) The averager ϕ^X approximates the value of each predicted successor state using nearby sample states $x \in X$. (e) The combination of the two averagers predicts stochastic transitions to sample states $x \in X$.

$a \in A$ that maximizes $\hat{Q}(s,a)$, its current estimate of the long-term value of each action. Conceptually, this estimate is the sum of an immediate reward and expected future rewards. AMBI estimates the immediate reward for each action directly using Equation 2. For the future rewards, it uses Equation 5 to approximate the distribution over successor states as a finite distribution over $X = \{s'_d \mid d \in D\}$, the set of all observed successor states. Since X is closed under all the actions, exact value iteration can compute the long-term value of each of these states.

During learning, the approximation described above can fail in one of two ways. First, the data D^a may not have enough data near s to approximate the effect of action a. Second, the sample X may not have enough states to approximate the successor states for some action. To achieve intelligent exploration, AMBI employs optimism to resolve both forms of uncertainty. It assumes the existence of two artificial states in X. The first artificial state, s^{opt} is an optimal state, from which every action deterministically earns reward $V^{\max}$, an upper bound on the value function, and transitions to s^{term}. The second artificial state, s^{term} represents all terminal states in the system. It is an absorbing state, so all actions from s^{term} earn zero reward and transition back to s^{term}. The known values of these artificial states are thus $\tilde{V}(s^{\text{term}}) = 0$ and $\tilde{V}(s^{\text{opt}}) = V^{\max}$.

The two artificial states are used to implement optimism in the face of uncertainty in the averagers ϕ^{S^a} and ϕ^X. AMBI defines these averagers to use Gaussian weighting, as follows. Without loss of generality, consider the averager $\phi^{X\cup\{s^{\text{opt}}\}}$, where $s^{\text{opt}} \notin X$. Given a state $s \in S$, define for each $x \in X$ the weight

$$w^X_{s,x} = e^{-\left(\frac{|x-s|}{b^X}\right)^2}, \tag{6}$$

where b^X is a parameter that controls the breadth of generalization over X and that corresponds to the standard deviation of the Gaussian kernel. The normalizer $Z^X_s = \sum_{x\in X} w^X_{s,x}$ corresponds to the weighted number of states in X used to approximate s and thus is a measure of the certainty of this approximation. If Z^X_s does not exceed a constant threshold c^X, then the averager approximates s with s^{opt}; otherwise, it approximates s with the weights $w^X_{s,x}$:

$$\phi_s^{X\cup\{s^{\text{opt}}\}}(x) = \begin{cases} \frac{w^X_{s,x}}{Z^X_s}, & \text{if } w^X_{s,x} \geq c^X \\ 0, & \text{if } w^X_{s,x} < c^X \end{cases} \tag{7}$$

$$\phi_s^{X\cup\{s^{\text{opt}}\}}(s^{\text{opt}}) = \begin{cases} 0, & \text{if } w^X_{s,x} \geq c^X \\ 1, & \text{if } w^X_{s,x} < c^X \end{cases} \tag{8}$$

The model-approximation averagers ϕ^{S^a} are defined in the same way. AMBI need not approximate the model at s^{term}, since actions are irrelevant at terminal states, but the averager for the value function should not approximate terminal states predicted by the model. The averager ϕ^X thus additionally specifies $\phi^X_{s^{\text{term}}}(s^{\text{term}}) = 1$ and $\phi^X_{s^{\text{term}}}(x) = 0$ for $x \neq s^{\text{term}}$.

Algorithm 1 Approximate Models Based on Instances

1: $X \leftarrow \{s^{\text{opt}}, s^{\text{term}}\}$ {Initialize state sample}
2: **for all** $a \in A$ **do** {Initialize experience sample}
3: $\quad D^a \leftarrow \{\langle s^{\text{opt}}, a, V^{\max}, s^{\text{term}}\rangle\}$
4: **end for**
5: $s \leftarrow$ initial state {Begin a trajectory}
6: $a \leftarrow \text{argmax}_a \left[\tilde{R}^a(s) + \sum_{x'\in X} \tilde{P}^a_s(x')\tilde{V}(x')\right]$
7: **while** $s \neq s^{\text{term}}$ **do**
8: $\quad$ Execute a
9: $\quad r \leftarrow$ observed reward
10: $\quad s' \leftarrow$ observed successor state
11: $\quad$ **if** s' is terminal **then**
12: $\quad\quad s' \leftarrow s^{\text{term}}$
13: $\quad$ **else**
14: $\quad\quad a' \leftarrow \text{argmax}_a \left[\tilde{R}^a(s) + \sum_{x'\in X} \tilde{P}^a_s(x')\tilde{V}(x')\right]$
15: $\quad\quad X \leftarrow X \cup \{s'\}$ {Update state sample}
16: $\quad$ **end if**
17: $\quad D^a \leftarrow D^a \cup \{\langle s, a, r, s'\rangle\}$ {Update experience sample}
18: $\quad$ Update ϕ^X and ϕ^{S^a} according to Equations 7 and 8
19: $\quad$ Update $\tilde{R}$ and $\tilde{P}$ according to Equations 2 and 5
20: $\quad$ Compute $\tilde{V}$ using value iteration
21: $\quad s \leftarrow s'$
22: $\quad a \leftarrow a'$
23: **end while**
24: **if** task is episodic **then**
25: $\quad$ Goto Line 5
26: **end if**

Algorithm 1 specifies the AMBI algorithm. It bears a high-level resemblance to Prioritized Sweeping [11], an algorithm that performs incremental updates to the estimated system model and value function after each unit of experience. The primary contribution of Prioritized Sweeping is an efficient method for propagating changes to the value function due to changes to the model, since the model updates for finite systems are trivial. AMBI adopts the prioritized value-function sweeps, but the primary computational burden stems from the updates to the derived model, due to generalization across states. Section 5.1 discusses implementation details that address computational efficiency.

AMBI also borrows the exploration mechanism of Prioritized Sweeping, which assumes transitions to an optimistic state until experience provides sufficient evidence to the contrary. The averager ϕ^{S^a} approximates the transition probabilty distribution function $\mathcal{P}^a_s$ at state s as $\mathcal{P}^a_{s^{\text{opt}}}$ if S^a does not contain enough states near s. AMBI also employs optimistic exploration in the approximation of the value function. After the approximate model predicts a possible transition to a state s', the averager ϕ^X approximates the value $V(s')$ of s' as $V(s^{\text{opt}})$ if X does not contain enough states near s'.

5. EXPERIMENTAL RESULTS

AMBI learns with good data efficiency by using a combination of model-based exploration, instance-based state representation, and averager-based approximation. This section describes experiments demonstrating that AMBI converges more rapidly to near-optimal policies than several other recent RL algorithms evaluated on some benchmark problems with continuous state spaces.

5.1 Implementation Details

A primary practical concern for an instance-based algorithm such as AMBI is computational complexity. After deriving the finite model $\tilde{M}$, fast tabular methods can reference it to update the value function $\tilde{V}$ efficiently using prioritized sweeps [11]. The computationally intensive steps of Algorithm 1 are Lines 7 and 8, which update the averagers ϕ^{S^a} and ϕ^X and the finite model $\tilde{M}$. In general, these steps require running time linear in the size of D, which is equal to the number of times the agent has acted.

The experimental implementation achieves a substantial reduction in the constant factor of this $O(D)$ running time by observing that the each newly sampled transition only changes the model appreciably in a local region of the state space. It sets the minimum nonzero value of $w^X_{s,x}$ to 0.01 in Equation 6 for both the model and value-function averagers. Thus the addition of a new transition from s only affects each averager ϕ at those states within distance $b\sqrt{-\log 0.01} = 2.146b$ from s. The implementation also prunes each averager ϕ so that the smallest nonzero value of ϕ_s is 0.01 (and renormalizes the remaining values), bounding to 100 the number of instances used to approximate s. Note that this pruning does not bias the approximation, which essentially becomes k-nearest neighbors with $k = 100$ and Gaussian weighting whenever sufficient data exists to override optimism. The precise thresholds used to prune did not significantly affect the performance of the algorithm.

With these two pruning mechanisms, $O(\log |D|)$ updates are possible by using a data structure such as a k-d tree to find the 100 nearest neighbors of each approximated state. For simplicity of implementation, the following sections describe results obtained with a simple linear-time binning approach that searched for neighbors in adjacent bins of width $2.146b$ for each averager. Due to memory and time constraints, this linear-time implementation stops adding new data after sampling 10000 transitions.

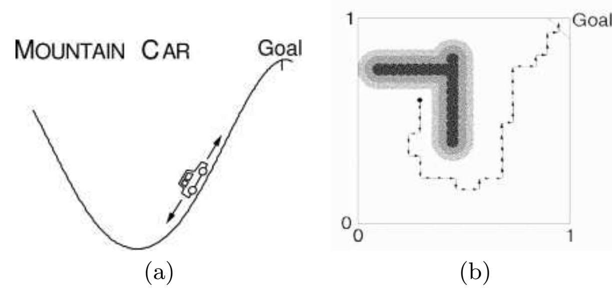


Figure 3: Two of the domains from the NIPS benchmarking workshop: (a) Mountain Car and (b) Puddle World.

5.2 Benchmark Performance

This section compares the performance of AMBI to algorithms submitted to the RL benchmarking workshop held at NIPS 2005 [4]. This event invited researchers to implement algorithms in a common interface for online RL. Participants computed their results locally, but direct comparisons are possible due to the standardized environment code, which presents the same sequence of initial states to each algorithm. Sections 5.2.1 and 5.2.2 examine two of the benchmark domains and give the AMBI parameters used to solve them. Section 5.2.3 evaluates the performance of AMBI against selected algorithms.

5.2.1 Mountain Car

In the Mountain Car simulation [16], an underpowered car must escape a valley (Figure 3a) by backing up the left slope to build sufficient energy to reach the top of the right slope. The agent has two state variables, horizontal position x and horizontal velocity v. The three available actions are `reverse`, `neutral`, and `forward`, which add -0.001, 0, and 0.001 to v, respectively. In addition, gravity adds $-0.0025\cos(3x)$ to v at each time step. The agent receives a reward of -1 for each time step before reaching the goal state. Episodes begin in a uniformly random initial position x and with $v = 0$, and they last for at most 300 time steps. The only domain knowledge available is the upper bound $V^{\max} = 0$ on the value function and the minimum and maximum values of each state variable: -1.2 and 0.5 for x and -0.07 and 0.07 for v.

AMBI scaled both state variables to $[0, 1]$. The generalization breadths were $b^{S^a} = 0.03$ to generalize the model and $b^X = 0.01$ to generalize the value function. Since Mountain Car is deterministic, the exploration thresholds were $c^{S^a} = 1$ and $c^X = 1$. To compute the value function, AMBI applied at most 1000 updates with minimum priority 0.01 after each transition.

5.2.2 Puddle World

The Puddle World [15] is a continuous grid world with the goal in the upper-right corner and two oval puddles (Figure 3b). The two state variables are the x and y coordinates, and the four actions correspond to the four cardinal directions. Each action moves the agent 0.05 in the indicated direction, with Gaussian noise added to each dimension with $\sigma = 0.01$. The agent receives a -1 reward for each action outside of the two puddles, with have radius 0.1 from two line segments, one from $(0.1, 0.75)$ to $(0.45, 0.75)$ and the

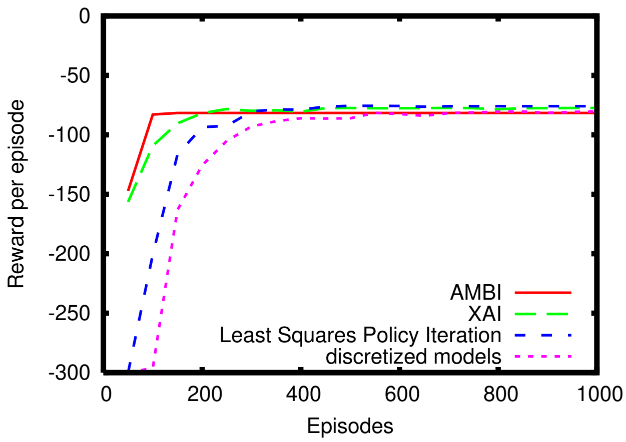


Figure 4: Learning curves for Mountain Car

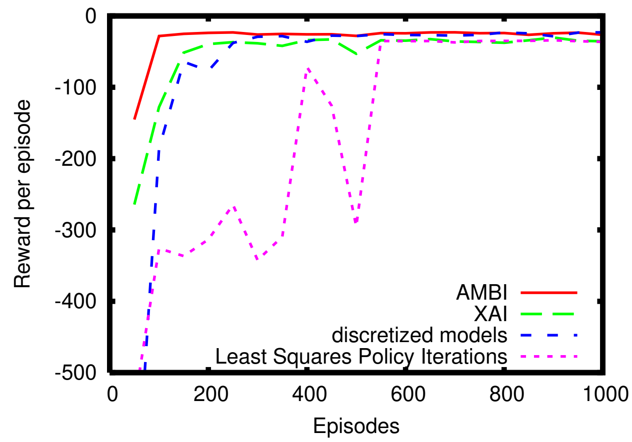


Figure 5: Learning curves for Puddle World

other from $(0.45, 0.4)$ to $(0.45, 0.8)$. Being in a puddle incurs a negative reward equal to 400 times the distance inside the puddle. The goal region satisfies $x + y \geq 0.95 + 0.95$.

For this domain, AMBI used generalization breadths $b^{S^a} = 0.05$ and $b^X = 0.02$. Although Puddle World is stochastic, thresholds $c^{S^a} = 1$ and $c^X = 1$ continued to suffice. AMBI used at most 1000 updates after each transition, with minimum priority 0.01.

5.2.3 *Benchmark Results*

Figures 4 and 5 compare the performance of AMBI to three selected algorithms. (Each point is the average of fifty sequential episodes, as reported to the NIPS workshop.) These three algorithms, implemented and parameterized by other researchers, were among the most competitive submitted. One is a model-based approach applied to a fixed discretization of the state space. This algorithm employed the same exploration mechanism as Prioritized Sweeping, but it lacked the instance-based representation and averager-based generalization of AMBI. Least Squares Policy Iteration [8] is similar to AMBI in that it uses a given sample of transitions to compute the parameters of a function approximator that best approximates the true value function. However, LSPI relies on random exploration and a fixed set of kernels to represent the state space. XAI (eXplore and Allocate, Incrementally) is a method that represents the value function with a network of radial basis functions, allocated online as the agent reaches unexplored regions of the state space [4]. It thus resembles AMBI in its instance-based use of Gaussian weighting for approximation, but XAI is a model-free method that uses gradient descent and Sarsa(λ) to update the value function. None of these algorithms achieves the same level of performance as AMBI, which combines instance-based state representation, averager-based generalization, and model-based exploration.

5.3 Ablation Study

This section illustrates the benefit of AMBI's approach to model-based RL in infinite systems. It compares three algorithms. The first is Prioritized Sweeping [11], a canonical model-based algorithm that reasons exactly in a fixed finite discretization of a given system. The second is a version of AMBI that uses the absolute model of sample transitions

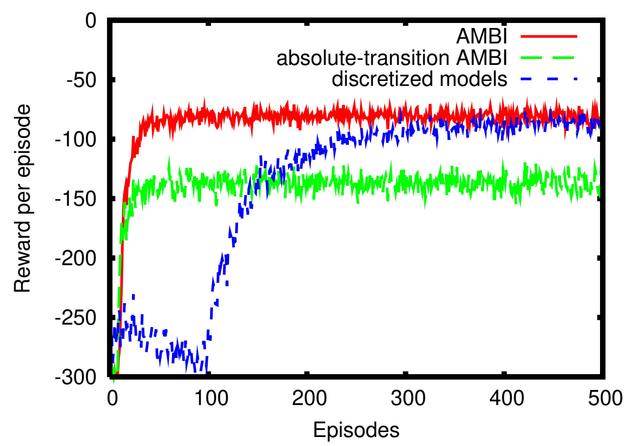


Figure 6: Learning curves for Mountain Car. Each curve is the average of 50 independent trials.

given in Equation 3, which in a sense is a more straightforward generalization of MDP transitions. The absolute model has the additional benefit that no value-function averager ϕ^X is necessary, since each successor state that the approximate transition functions $\hat{P}^a$ propose exists in X. This version of AMBI can also be construed as an online implementation of Kernel-Based Reinforcement Learning [12] (KBRL), discussed in Section 6, combined with the exploration mechanism of Prioritized Sweeping. The third algorithm is AMBI, which employs the relative model of sample transitions given in Equation 4 and necessarily a value-function averager.

Figure 6 shows the performance of each algorithm, averaged over 50 independent trials in the Mountain Car domain. This implementation of Prioritized Sweeping uses the same parameters as the finite model-based algorithm submitted to the NIPS workshop: it discretizes each state dimension into 100 intervals and uses $n_{\text{known}} = 1$. AMBI used the same parameters described in Section 5.2.1.

Absolute-transition AMBI converges much more quickly than discrete Prioritized Sweeping, but at the expense of converging to suboptimal policies. Further experimentation has shown that decreasing b^{S^a} improves the average quality of the final policy but quickly decreases the learning speed of the algorithm. The standard version of AMBI uses the more accurate relative transition generalization to preserve fast convergence while achieving near-optimal policies in this domain. For comparison, Figure 7 illustrates typical learned policies for both versions of AMBI. An optimal policy would execute `forward` roughly when the velocity is positive, in the upper half of the state-space diagram, and it would execute `reverse` roughly when the velocity is negative, in the lower half of the state-space diagram. This run of absolute-transition AMBI incorrectly selects `reverse` in a large region with positive velocity. Inspection of the relevant states revealed that the local neighborhood of the sample $S^{\texttt{reverse}}$ happened to contain more high-value states. The absolute transition model incorrectly concluded that the `reverse` action would transition to this higher-value region; the relative transition model correctly concluded that this action decreases the value of any state in the neighborhood.

6. DISCUSSION AND RELATED WORK

The primary contribution of this paper is its integration of model-based reasoning with stable function approximation. AMBI extends the data efficiency of model-based methods to continuous systems, which previously presented the difficulty of representing continuous models. Atkeson, Moore, and Schaal addressed this problem in the deterministic case, also using locally weighted learning from instances [1]. Their application of locally weighted regression estimated the average successor state for each state-action pair; AMBI approximates the distribution over successor states and thus copes with forms of stochasticity beyond simple noise. Atkeson et al. also did not address the issue of exploration in continuous systems. AMBI permits the application of intelligent exploration mechanisms originally designed for finite systems. It employs the same mechanism as Prioritized Sweeping [11] and R-MAX [3], perhaps opening the door for generalizing the latter algorithm's polynomial-time PAC convergence guarantees to certain continuous systems.

Introducing model-based reasoning to function approximation also provides novel insight into the problem of generalizing from finite data to knowledge of an infinite system. Most approaches to function approximation rely on a static scheme for generalizing the value function directly, despite the difficulty in intuiting the structure of value functions. AMBI explicitly generalizes first in a model of the system, where intuitions may be easier to represent. For example, a high degree of generalization is possible in the model for Mountain Car, since the effect of an action changes smoothly with the current state. In contrast, the optimal value function for this system includes large discontinuities in locations that are impossible to predict without first knowing the optimal policy: the discontinuity separates those regions of the state space where the agent has sufficient energy to escape the valley and from those regions where it must first build energy. Approaches that only generalize the value function must use little enough generalization to represent this discontinuity accurately; AMBI uses a learned model to generalize both broadly and accurately.

Mahadevan also uses a learned model to improve value-function approximation, by analyzing state-space topology to compute "proto-value functions" that form a linear basis for the learned value function [10]. This work separates learning into two phases, one that gathers data on the state space and one that learns the weights for the proto-value functions without any further model-based reasoning. Glaubius and Smart also analyze state-space topology, to discover overlapping low-dimensional manifolds over which to learn the value function [5]. Both of these methods only employ models to determine generalization schemes for model-free learning algorithms that do not address the problem of efficient exploration.

Similarly, although other algorithms use instance-based representations for stable function approximation, they typically do not benefit from exploration methods that arise from model-based reasoning. For example, Interpolative Function Approximator based Q-Learning also builds upon Gordon's convergence proofs for averagers, defining a model-free algorithm that can converge to the optimal value function in the limit [17]. Like the original Q-learning algorithm, this algorithm relies on a given exploration policy to visit the appropriate regions of the state space sufficiently.

Kernel-Based Reinforcement Learning also converges to the optimal value function in the limit, applying an approximate form of value iteration to a set of instance states [12]. It defines an approximate Bellman equation that implicitly uses the absolute-transition model defined in Equation 3, but it does not benefit from any explicit model-based reasoning. KBRL is also essentially an offline algorithm that simply computes a value function from a sample, without addressing exploration. Section 5.3 discusses experimental results using an online adaptation of KBRL that incorporates model-based exploration. Some preliminary experiments also applied KBRL to Mountain Car, recomputing a value function from cumulative data after each episode. These experiments showed that random exploration proved ineffective, almost never even finding the goal state.

One noteworthy limitation of AMBI is its scalability. As an instance-based algorithm, its time and space complexity grow with the amount of data it collects. The implementation used in Section 5 simply stops adding data after reaching a fixed threshold, but a more principled approach would strive to keep the most useful samples without introducing significant bias. AMBI could stop adding states to X if the model grows too large to update the value function efficiently, and it can independently stop adding transitions to the D^a if the sample grows too large update the model efficiently.

AMBI is also vulnerable to the curse of dimensionality. As the dimensionality grows, exponentially more data is required to explore each neighborhood of the state space. One solution is to select or to learn more sophisticated kernels that permit generalization of a stored transition beyond its local neighborhood. For example, in real-world domains, many actions are independent of some subset of the state dimensions, and the approximation of those actions can thus generalize freely over those dimensions. Using the relative-transition model may also play an important role here, since the algorithm must learn that an action leaves an independent dimension unchanged, instead of changing it to a previously observed value.

7. CONCLUSION

Reinforcement learning in infinite systems requires accurate generalization from finite data, but standard approaches only apply generalization directly to the value function. Many systems of interest exhibit more intuitive structure in their

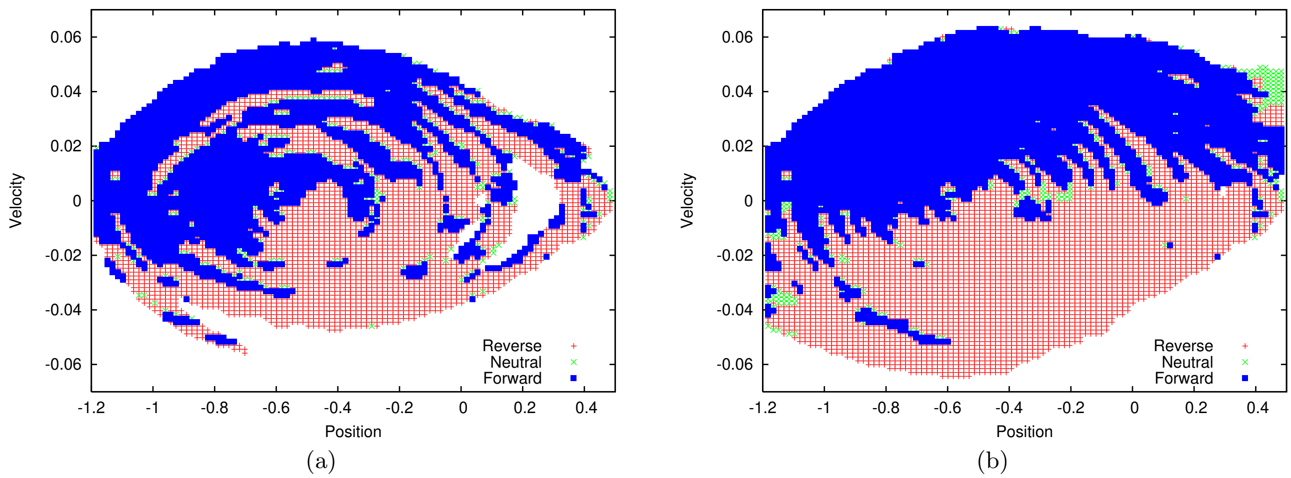


Figure 7: Mountain-Car policies learned using (a) absolute-transition AMBI and (b) standard AMBI. The solid region of the state space indicates where the policy selects the `forward` action; the hatched region indicates where it selects the `reverse` action.

one-step dynamics than in the optimal value function. This observation suggests a model-based solution that generalizes first from data to a model. The AMBI algorithm employs averagers both to approximate the model and the value function. It derives a finite representation of the system that both allows efficient planning and intelligent exploration. These attributes allow AMBI to learn some standard benchmark systems more efficiently than many contemporary RL algorithms.

Acknowledgments

We would like to thank Michael Littman and the Rutgers Laboratory for Real-Life Reinforcement Learning for valuable discussions. Shimon Whiteson, Matt Taylor, and anonymous reviewers provided helpful comments and suggestions. This research was supported in part by NSF CAREER award IIS-0237699 and DARPA grant HR0011-04-1-0035.

8. REFERENCES

[1] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113, 1997.

[2] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, 1995.

[3] R. I. Brafman and M. Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 953–958, 2001.

[4] A. Dutech, T. Edmunds, J. Kok, M. Lagoudakis, M. Littman, M. Riedmiller, B. Russell, B. Scherrer, R. Sutton, S. Timmer, N. Vlassis, A. White, and S. Whiteson. Reinforcement learning benchmarks and bake-offs II. http://www.cs.rutgers.edu/~mlittman/topics/nips05-mdp/bakeoffs05.pdf, 2005.

[5] R. Glaubius and W. D. Smart. Manifold representations for value-function approximation. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004.

[6] G. J. Gordon. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[7] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 260–268, 1998.

[8] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[9] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 1995.

[10] S. Mahadevan. Samuel meets Amarel: Automating value function approximation using global state space analysis. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005.

[11] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[12] D. Ormnoneit and Ś. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2):161–178, Nov. 2002.

[13] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[14] M. Riedmiller. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the European Conference on Machine Learning*, 2005.

[15] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, 1996.

[16] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[17] C. Szepesvári and W. D. Smart. Interpolation-based Q-learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[18] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), Mar. 1995.