To appear in *The Seventh Symposium on Abstraction, Reformulation, and Approximation* (SARA 07), Whistler, Canada, July 2007.

Model-Based Exploration in Continuous State Spaces

Nicholas K. Jong and Peter Stone

The University of Texas at Austin, Austin TX 78712, USA,
{nkj,pstone}@cs.utexas.edu,
WWW home page: http://www.cs.utexas.edu/users/{nkj,pstone}

Abstract. Modern reinforcement learning algorithms effectively exploit experience data sampled from an unknown controlled dynamical system to compute a good control policy, but to obtain the necessary data they typically rely on naive exploration mechansisms or human domain knowledge. Approaches that first learn a model offer improved exploration in finite problems, but discrete model representations do not extend directly to continuous problems. This paper develops a method for approximating continuous models by fitting data to a finite sample of states, leading to finite representations compatible with existing model-based exploration mechanisms. Experiments with the resulting family of fitted-model reinforcement learning algorithms reveals the critical importance of how the continuous model is generalized from finite data. This paper demonstrates instantiations of fitted-model algorithms that lead to faster learning on benchmark problems than contemporary model-free RL algorithms that only apply generalization in estimating action values. Finally, the paper concludes that in continuous problems, the exploration-exploitation tradeoff is better construed as a balance between exploration and generalization.

1 Introduction

Reinforcement learning (RL) algorithms must balance two motives in selecting actions in controlled systems: exploration and exploitation [1]. Exploratory actions attempt to gather useful data about the system; exploitative actions attempt to maximize expected rewards given the data. Effective exploration remains a challenging research problem, with many RL implementations relying on relatively naive approaches. For example, the seminal Q-learning algorithm [2], which underlies a large fraction of ongoing RL research and most current RL applications, promises asymptotic convergence to an optimal control policy, given an exploration policy that attempts every action in every state infinitely often. In practice, most implementations explore by relying on random deviations from the learned policy. Any sequence of actions is possible in such a scheme, but a sequence becomes exponentially unlikely the longer it deviates from the learned policy. Such inefficient exploration methods help to explain the limitations of applying RL methods to real-world problems, which demand fast convergence to reasonable policies in large or infinite state spaces.

Model-based approaches to RL facilitate more informed exploration by explicitly estimating the dynamics of the system before attempting to estimate the optimal policy. Uncertainty in the learned model can direct the learning algorithm to seek the data most likely to improve exploitation. Such approaches led to the first probabilistic convergence guarantees to near-optimal policies with finite amounts of data [3]. However, this body of research usually relies on tabular representations of models that presuppose discrete problems; model learning for continuous problems has been restricted to the case of deterministic dynamics [4]. Partly for this reason, state-of-the-art algorithms for continuous problems, such as LSPI [5], rely on model-free techniques, which approximate the long-term value of each action in every state directly from data. However, these algorithms still rely on random exploration to acquire this data. To make matters worse, many modern algorithms employ computationally expensive supervised learning mechanisms, which permit them to update their exploration policies very intermittently. In contrast, model-based algorithms typically support incremental updates that permit immediate changes to the exploration policy in response to each new piece of data.

This paper develops model-based algorithms suitable for continuous problems. It addresses the question of how to represent the transition model for an action, which must specify for infinitely many states a successor state distribution over an infinite set. The successor state distribution may be approximated with a finite sample generalized from the data, but this distribution must still be parameterized by the infinite state set. However, if this transition model is used with fitted value iteration [6], an algorithm for computing policies in infinite problems using a finite state sample, then it suffices to represent the transition model explicitly at only a finite number of points. The resulting fitted model permits the application of the simple but effective exploration mechanism from R-MAX [7], a model-based algorithm designed for finite problems.

2 Background

Most RL algorithms assume that the controlled system constitutes a Markov decision process (MDP) [8]. An MDP $\langle S, A, \mathcal{P}, \mathcal{R} \rangle$ comprises a set of states S, a finite set of actions A, a transition function $\mathcal{P} : S \times A \to \Delta S$, and a reward function $\mathcal{R} : S \times A \to \mathbb{R}$. For all $s \in S$ and $a \in A$, $\mathcal{P}(s, a) = \mathcal{P}_{sa}$ gives the probability density function over successor states s' given that action a is executed in state s, so $\Pr(s'|s, a) = \mathcal{P}_{sa}(s')$. The reward function gives the expected reward $E[r|s, a] = \mathcal{R}(s, a) = \mathcal{R}_{sa}$ for executing action a in state s. Finally, MDPs satisfy the Markov assumption, which states that the successor state s' and the reward r depend only on s and a. In other words, s' and r are conditionally independent of all other variables given s and a.

In this paper, it will be necessary to reason about the composition of Markovian transition functions, similar to the composition of MDPs in [6]. To this end, suppose $f : Y \to \Delta X$ and $g : Z \to \Delta Y$ are transition functions from Y to X and from Z to Y, respectively. Then the composition of f and g, $f \circ g : Z \to \Delta X$ is obtained by marginalizing over the values of Y:

$$(f \circ g)_z(x) = \Pr(x|z) = \int_y \Pr(x \wedge y|z)\,\mathrm{d}y = \int_y \Pr(x|y,z)\Pr(y|z)\,\mathrm{d}y \tag{1}$$

$$= \int_y \Pr(x|y)\Pr(y|z)\,\mathrm{d}y \tag{2}$$

$$= \int_y f_y(x) g_z(y)\,\mathrm{d}y, \tag{3}$$

where (2) follows from the conditional independence of x from z given y.[1] Note that for the finite case, the composition of transition functions corresponds directly to the multiplication of the appropriate transition matrices.

The optimal value function $V : S \to \mathbb{R}$ for an MDP specifies the maximum possible expected cumulative reward $V(s)$ given optimal behavior and starting from state $s \in S$. This value function satisfies the Bellman optimality equations: for all $s \in S$,

$$V(s) = \max_{a \in A}\left[\mathcal{R}_{sa} + \int_{s' \in S} \mathcal{P}_{sa}(s')V(s')\,\mathrm{d}s'\right]. \tag{4}$$

(A discount factor $\gamma \in [0,1]$ may be used to ensure that this system has a solution.) Given V, an optimal policy $\pi : S \to A$ may be defined by $\pi(s) = \mathrm{argmax}_{a \in A} Q(s,a)$, where $Q(s,a) = \mathcal{R}_{sa} + \int_{s' \in S} \mathcal{P}_{sa}(s')V(s')\,\mathrm{d}s'$.

3 Model Approximation

Model-based RL algorithms estimate the transition and reward functions $\mathcal{P}$ and $\mathcal{R}$ from experience data. They can then use these estimates relatively directly with (4) to compute the optimal value function and policy. For concreteness, let $s_0, a_0, r_1, s_1, \ldots, r_t, s_t$ be the data, with t being the current time step. In episodic tasks, a special state $s^{\text{terminal}} \notin S$ designates the end of an episode. If $s_i = s^{\text{terminal}}$, then a_i and r_{i+1} are undefined, and s_{i+1} is the initial state in the next episode. Additionally, define the set of transition instances $D = \left\{i \mid 0 \le i < t \wedge s_i \ne s^{\text{terminal}}\right\}$. For convenience, we also define subsets of D that condition the data on specific actions and states. Let $D^a = \{i \in D \mid a_i = a\}$ be the set of instances that match action a, let $D^a_s = \{i \in D^a \mid s_i = s\}$ be the set of instances that also matches state s, and let $D^a_{ss'} = \{i \in D^a_s \mid s_{i+1} = s'\}$ be the set of instances that also matches successor state s'.

For finite MDPs, straightforward maximum likelihood estimation of $\mathcal{P}$ and $\mathcal{R}$ is both simple and effective. The model may be computed from D as $\hat{\mathcal{P}}_{sa}(s') = \frac{|D^a_{ss'}|}{|D^a_s|}$ and $\hat{\mathcal{R}}_{sa} = \frac{\sum_{i \in D^a_s} r_{i+1}}{|D^a_s|}$. Initially, D^a_s will be quite small everywhere, but by

[1] $f \circ g$ is not strictly a Markovian transition function, since x is not conditionally independent of y given z, but this subtlety is not relevant to the results of this paper.

the pigeonhole principle the estimate will become quite reliable at some state-action sa. Reliable regions of the model enable a model-based exploration mechanism to direct the agent to regions of the state space where more data is needed, until adequate data exists to estimate the model at every reachable state. This approach to exploration in finite problems is precisely the one taken explicitly in E^3 [3] and implicitly in prioritized sweeping [9] and R-MAX [7].

In very large finite MDPs, the updating the model for only one state-action at a time may require too much data in practice. In infinite MDPs, the algorithm may never visit the same state twice, precluding accurate estimation of the model parameters entirely. One of the primary contributions of this paper is a robust method for approximating the maximum likelihood model from data. The method decomposes the estimated transition function into the composition of components that can be computed easily from the data. A key feature of the final method will be that one of these components generalizes the model across nearby states, but for the sake of clarity the initial description of the decomposition will address the case without generalization.

3.1 Decomposition of the Transition Function

Consider the task of estimating the effect of executing action a in state s, given data D. Instead of directly estimating the transitions as a probability distribution over S as a function of $S \times A$, define a two-stage transition function that first transitions from state-action sa to a state-instance $si \in S \times D$, then from state-instance si to a successor state $s' \in S$. A dynamic Bayesian network of this formulation appears in Fig. 1.

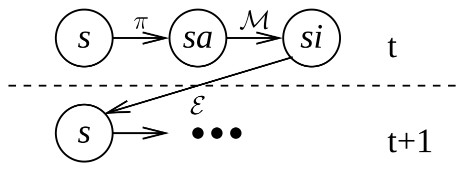


Fig. 1. Dynamic Bayesian network showing the decomposition of the transition function for an approximated MDP. The policy π determines the conditional distribution of sa given s. The model instance transition function $\mathcal{M}$ determines the conditional distribution of si given sa. The action effect transition function $\mathcal{E}$ determines the conditional distribution of s at time $t+1$ given si from time t.

A model instance transition function $\mathcal{M} : S \times A \rightarrow \Delta(S \times D)$ maps each state-action sa to a state-instance si with probability $\mathcal{M}_{sa}(s, i)$. Intuitively, $\mathcal{M}$ replaces the action component of sa with a specific instance i from the agent's experience that represents the predicted effect of a. The state-instance si implies that the same thing that happened at time step i will happen again, this time at state s. This transition function thus accounts for the stochasticity in the domain, by replacing the potentially stochastic action a with a specific outcome i. Note that $\mathcal{M}$ preserves the value of the state s when it transitions a state-action sa to a state-instance si.

In the absence of generalization, the agent has no reason to believe that the action effect at instance i will recur at state s unless $s_i = s$ and $a_i = a$. Hence, the exact model instance transition function is

$$\mathcal{M}_{sa}^{\text{exact}}(s, i) \propto \delta_{ss_i}\delta_{aa_i}, \tag{5}$$

where $\delta_{xy} = 1$ if $x = y$ and 0 otherwise. In the same vein, given state-instance si and $s = s_i$, it must be the case that $s' = s_{i+1}$, since the transition to si accounted for any nondeterminism. The absolute effect transition function $\mathcal{E}_{si}^{\text{abs}} : S \times D \to \Delta S$ reflects this expectation:

$$\mathcal{E}_{si}^{\text{abs}}(s') = \delta_{s's_{i+1}}. \tag{6}$$

It can be verified that the composition of $\mathcal{E}_{si}^{\text{abs}}$ and $\mathcal{M}_{sa}^{\text{exact}}$ yields the maximum likelihood estimator for $\mathcal{P}$ given above:

$$\begin{aligned}\left(\mathcal{E}^{\text{abs}} \circ \mathcal{M}^{\text{exact}}\right)_{sa}(s') &= \sum_{i \in D} \delta_{s's_{i+1}} \frac{\delta_{ss_i}\delta_{aa_i}}{\sum_{i \in D}\delta_{ss_i}\delta_{aa_i}} \\ &= \sum_{i \in D} \frac{\delta_{s's_{i+1}}\delta_{ss_i}\delta_{aa_i}}{|D_s^a|} \\ &= \frac{|D_{ss'}^a|}{|D_s^a|} \\ &= \hat{\mathcal{P}}_s^a(s').\end{aligned}$$

3.2 Model Generalization

The preceding section presented a novel computation of the exact maximum likelihood estimator for $\mathcal{P}$, but as discussed at the beginning of Sect. 3, in very large or infinite MDPs this estimator is impractical. This section describes alternative definitions of the model instance transition function $\mathcal{M}$ and action effect transition function $\mathcal{E}$ that are more useful in continuous problems. To predict the effects of actions at infinitely many states given only finite data, the learned model must use some form of generalization and hence inductive bias. This section of the paper places additional assumptions on the state space of the MDP to be learned. In particular, it assumes the state space is a bounded subset of some Euclidean space, and it assumes that nearby states tend to induce similar dynamics and reward for each action.

The approximate model lifts the restriction that the exact model imposes, allowing a state-action sa to transition to a state-instance si such that $s_i \neq s$. However, the model weights each transition according to a decreasing function of the euclidean distance $|s_i - s|$ between s_i and s. This paper uses Gaussian weighting:

$$\mathcal{M}_{sa}^{\text{approx}}(s, i) \propto \delta_{aa_i} e^{-\left(\frac{|s - s_i|}{b}\right)^2}, \tag{7}$$

where b is a parameter that controls the breadth of generalization across the state space. This parameter critically affects learning performance. A large degree of generalization permits very rapid learning, but in some cases overgeneralization can prevent the algorithm from ever finding a good policy.

Even moderate amounts of generalization suggest a modification to the absolute action effect transition function $\mathcal{E}^{\text{abs}}$ defined in Sect. 3.1, as shown in Fig. 2. For a given state-instance si, predicted the successor state to be s_{i+1}, the actual successor state for the instance i, makes less sense the farther s_i is from s. Early experiments demonstrated that the breadth of generalization can be quite large relative to the distance traveled in one time step. Otherwise, the amount of data required may be prohibitive, despite potential regularities in the system's dynamics. For example, a mobile robot whose state includes its pose should be able to generalize its action model over large regions of free space. The relative action effect transition function thus attempts to isolate for a given instance i the contribution of the action a_i from the contribution of the state s_i on the successor s_{i+1}:

$$\mathcal{E}^{\text{rel}}_{si}(s') = \begin{cases} 1, \text{if } s_{i+1} = s^{\text{terminal}} \wedge s' = s^{\text{terminal}} \\ 1, \text{if } s' = s + (s_{i+1} - s_i) \\ 0, \text{otherwise.} \end{cases} \tag{8}$$

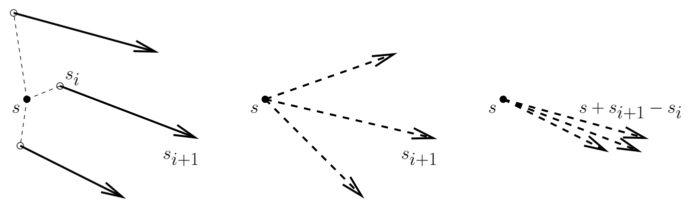


Fig. 2. The action effect transition function $\mathcal{E}$ can determine the fidelity of the approximated model to the true dynamics. (a) Approximating the effect of some action a at a given state s using three nearby instances. (b) Absolute action effects predict transitions to the exact successor states previously visited. (c) Relative action effects better capture the dynamics of the system by applying the appropriate vectors to the present state s.

Finally, the approximation of the reward function $\mathcal{R}^{\text{approx}}$ is similar to the approximation of the model approximation transition function. For a given state-action sa, the approximated expected reward is a weighted average of the rewards for the instances used to approximate a near s:

$$\mathcal{R}^{\text{approx}}_{sa} = \sum_{i \in D} r_{i+1} \mathcal{M}^{\text{approx}}_{sa}(s, i). \tag{9}$$

4 Fitted-Model Learning Algorithms

Section 3 gave an approximation of the transition and reward functions for an unknown continuous-state MDP, but a complete model-based algorithm also requires a practical method for computing the value function from the model and an exploration mechanism. This section integrates the contributions of Sect. 3 with existing algorithms that play each of these roles.

4.1 Fitted Models

The approximate model instance transition function $\mathcal{M}^{\text{approx}}$ induces a continuous MDP $\langle S, A, \mathcal{P}^{\text{approx}}, \mathcal{R}^{\text{approx}} \rangle$, with $\mathcal{P}^{\text{approx}} = \mathcal{E}^{\text{rel}} \circ \mathcal{M}^{\text{approx}}$ and $\mathcal{R}^{\text{approx}}$ given by (9). Computing the optimal value function for even this approximate model is impossible in general, since the transition and reward functions still vary continuously over the infinite state-action space $S \times A$.

Fitted value iteration [6] provides an algorithm for approximating the optimal value function of continuous-state MDPs. The algorithm uses a finite sample $X \subset S$ of states to represent the value function, for an arbitrary value function approximation scheme that represents the value of any state $s \in S$ as some weighted average of the values of X. That is, the function approximator must compute the value of a state $s \in S$ as $V(s) = \sum_{x \in X} \mathcal{F}_s(x) V(x)$, where $\sum_{x \in X} \mathcal{F}_s(x) = 1$ and $\mathcal{F}_s(x) \geq 0$. In other words, $\mathcal{F} : S \to \Delta X$ must be a transition function that transitions every state in S to one of the states in the finite sample X. Then interleaving steps of value iteration with fitting the value function to the function approximation scheme is equivalent to applying standard value iteration [10] to the derived MDP $\langle X, A, \mathcal{F} \circ \mathcal{P}, \mathcal{R} \rangle$, as diagrammed in Fig. 3 and 4. This equivalence ensures that function approximation does not cause the value function computation to diverge.

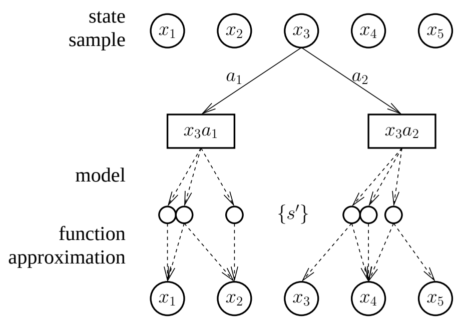


Fig. 3. This diagram shows a continuous MDP with two actions being fitted to a state sample of size 5. For clarity, only the transitions for state s_3 are shown.

To compute a value function for the approximate model defined in (7) and (9), it suffices to substitute $\mathcal{P}^{\text{approx}}$ for $\mathcal{P}$ and $\mathcal{R}^{\text{approx}}$ for $\mathcal{R}$ in fitted value iteration.

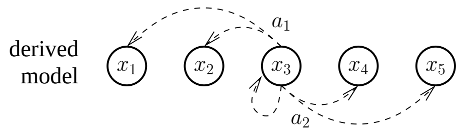


Fig. 4. This diagram shows the finite MDP derived from fitting the continuous MDP in Fig. 3.

Approximating the value function for the learned model is thus equivalent to computing the exact value function for the finite MDP $\langle X, A, \mathcal{F} \circ \mathcal{E} \circ \mathcal{M}, \mathcal{R} \rangle$. Fig. 5 illustrates this decomposition.

Many function approximation schemes are possible for choosing X and defining $\mathcal{F}$. In all the experiments described in this paper, X is a uniform grid spanning the state space, and $\mathcal{F}_{s'}(x)$ gives the coefficients for multilinear interpolation of s' from the 2^d corners of the hypercube containing s', where d is the dimensionality of the state space. Preliminary experiments showed that this simple function approximation scheme performed better than a number of alternatives, including instance-based approaches that added either visited states s_t to X or predicted successors s' to X as necessary.

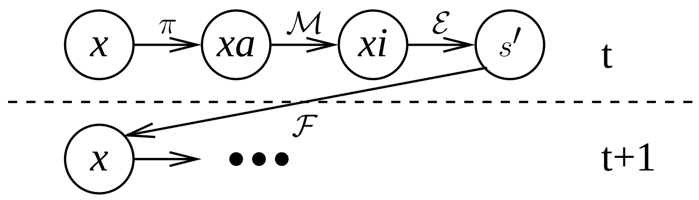


Fig. 5. Dynamic Bayesian network showing the decomposition of transitions in the derived MDP, solved using standard value iteration, into components of the model approximation.

4.2 Fitted R-max

Section 4.1 showed how to estimate the optimal value function from data by first approximating a model, but one of the primary motivations behind extending model-based methods to continuous problems is to take advantage of intelligent exploration methods. This section describes one simple but effective model-based algorithm for finite problems and shows how to incorporate its exploration mechanism into fitted value iteration.

R-MAX is a relatively simple model-based algorithm that implements a standard principle of exploration: optimism in the face of uncertainty [7]. It maintains maximum-likelihood estimates of the model parameters $\hat{\mathcal{P}}_{sa}(\cdot)$ and $\hat{\mathcal{R}}_{sa}$ for every state s and action a, but it only employs these estimates given sufficient data to have confidence in their accuracy. Let $n(s, a)$ denote the number of times action a has been executed in state s. Then R-MAX estimates the value function using

$$\hat{Q}(s, a) = \begin{cases} V^{\max} \text{ if } n(s, a) < m \\ \hat{\mathcal{R}}_{sa} + \sum_{s' \in S} \hat{\mathcal{P}}_{sa}(s')\hat{V}(s') \text{ if } n(s, a) \geq m \end{cases} \tag{10}$$

where $\hat{V} = \max_{a \in A} \hat{Q}(s, a)$, $V^{\max}$ is an upper bound on the value function,[2] and m is a constant. The modified Bellman equations can still be solved using a standard MDP planning algorithm, such as value iteration.

The optimistic value function explicitly rewards the algorithm for executing actions in uncertain states. The parameter m determines the amount of exploration required before the algorithm is certain about the effects of a state-action pair. Furthermore, augmenting the value function in this manner causes the agent to seek out states that are either actually high in value or where "exploration bonuses" are available for executing unfamiliar state-actions. [11] showed how the exploration threshold m relates to the likely error in the estimates $\hat{\mathcal{P}}$, leading to bounds on the amount of exploration required before converging to a probably approximately optimal policy.

Although the use of generalization in approximating a fitted model eliminates such guarantees of convergence to optimal behavior, the exploration mechanism of R-MAX can still be applied to fitted value iteration simply by substituting in the parameters of the derived finite MDP into (10) and appropriately defining an approximation of $n(s, a)$. This latter quantity denotes the number of times a was executed in state s, so the logical analog is the sum of the unnormalized kernel values used to weight the transitions from a state s to each $i \in D$:

$$\tilde{n}(s, a) = \sum_{i \in D} \delta_{aa_i} e^{-\left(\frac{|s - s_i|}{b}\right)^2}. \tag{11}$$

Thus $n(s, a)$ now counts both the actual data for a at s as well as "partial" data generalized from executions of a near s. The fitted R-MAX algorithm thus computes the following value function:

$$Q(s, a) = \begin{cases} V^{\max} \text{ if } n(s, a) < m \\ \mathcal{R}^{\text{approx}}_{sa} + \sum_{x' \in X} \left(\mathcal{F} \circ \mathcal{E}^{\text{relative}} \circ \mathcal{M}^{\text{approx}}\right)_{sa}(x')V(x') \text{ if } n(x, a) \geq m \end{cases} \tag{12}$$

At each time step, fitted R-MAX adds the just observed transition to D, updates $\mathcal{M}^{\text{approx}}$ and $\mathcal{F}$, and then applies value iteration to solve (10). The algorithm then behaves greedily with respect to $Q(s_t, \cdot)$. In practice, the composition $\tilde{\mathcal{P}} = \mathcal{F} \circ \mathcal{E}^{\text{relative}} \circ \mathcal{M}^{\text{approx}}$ need not be recomputed after each time step. By caching the appropriate intermediary values, the finite transition function $\tilde{\mathcal{P}} : X \times A \to \Delta X$ can be repaired to reflect each new instance i. In the same spirit, prioritized sweeping [9] can be used to update the value function efficiently to reflect changes in the approximate model.

[2] [7] reasons with $R_{\max}$, an upper bound on the one-step reward, since its version of the algorithm computes finite-horizon value functions.

5 Experimental Results

Fitted R-MAX learns with good data efficiency by using a combination of model-based exploration and stable function approximation. This section describes experiments demonstrating that fitted R-MAX converges more rapidly to near-optimal policies than several other recent RL algorithms evaluated on some benchmark problems with continuous state spaces. It then examines the importance of the relative action effect transition function $\mathcal{E}^{\text{rel}}$ compared to the absolute version $\mathcal{E}^{\text{abs}}$. Finally, it investigates the importance of the generalization breadth parameter, b.

5.1 Implementation Details

A primary practical concern for any instance-based algorithm is computational complexity. The computationally intensive step of fitted R-MAX is the incremental update to the derived finite model. 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 the (unnormalized) Gaussian weighting to 0.01 in (7). Thus the addition of a new transition from s only affects those sample states $x \in X$ 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 $\mathcal{M}_{sa}^{\text{approx}}(s, i)$ 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.

5.2 Benchmark Performance

This section compares the performance of fitted R-MAX to algorithms submitted to the RL benchmarking workshop held at NIPS 2005 [12]. 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. This sections examines two of the benchmark domains and gives the fitted R-MAX parameters used to solve them. It then evaluates the performance of fitted R-MAX against selected algorithms.

Mountain Car In the Mountain Car simulation [1], an underpowered car must escape a valley (Fig. 6a) 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.

Fitted R-MAX scaled both state variables to $[0, 1]$. The generalization breadth b was 0.08. X consisted of uniform 64×64 grid overlaying the state space. Since Mountain Car is deterministic, the exploration thresholds was $m = 1$. To compute the value function, fitted R-MAX applied at most 1000 updates with minimum priority 0.01 after each transition.

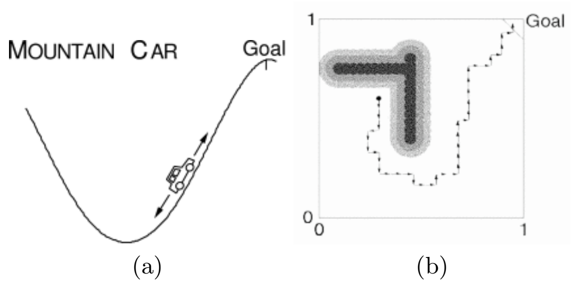
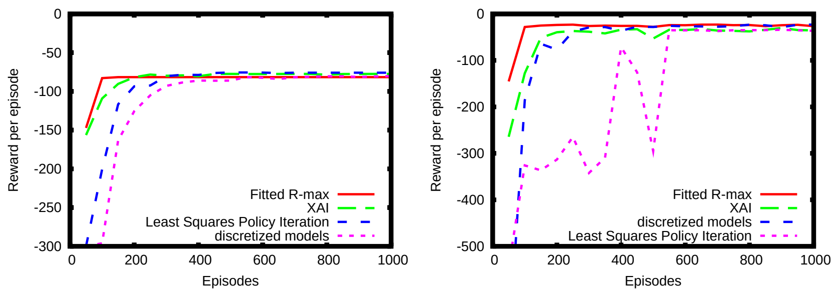


Fig. 6. Two of the domains from the NIPS benchmarking workshop: (a) Mountain Car and (b) Puddle World.

Puddle World The Puddle World [13] is a continuous grid world with the goal in the upper-right corner and two oval puddles (Fig. 6b). 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 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, fitted R-MAX used generalization breadth $b = 0.08$. A 64×64 grid was again used for X. Although Puddle World is stochastic, thresholds $m = 1$ continued to suffice. Fitted R-MAX used at most 1000 updates after each transition, with minimum priority 0.01.

Benchmark Results Figure 7 compares the performance of fitted R-MAX 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 fitted R-MAX. Least Squares Policy Iteration [5] is similar to fitted R-MAX 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 [12]. It thus resembles fitted R-MAX 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 fitted R-MAX, which combines instance-based model approximation, stable function approximation, and model-based exploration.



(a) (b)

Fig. 7. Learning curves for (a) Mountain Car and (b) Puddle World

5.3 Ablation Study

This section illustrates the benefit of fitted R-MAX's approach to model-based RL in infinite systems. It compares three algorithms. The first is fitted R-MAX, employing the relative action effect transition function $\mathcal{E}^{\mathrm{rel}}$ given in (8). The second is a version of fitted R-MAX that uses the absolute action effect transition function $\mathcal{E}^{\mathrm{abs}}$ given in (6), to measure the importance of action effect component of the transition function. The third algorithm is the original discrete R-MAX algorithm [7], to measure the importance of the novel decomposition of the transition function.

Figure 8 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 $m = 1$. Fitted R-MAX used the same parameters described in Sect. 5.2.

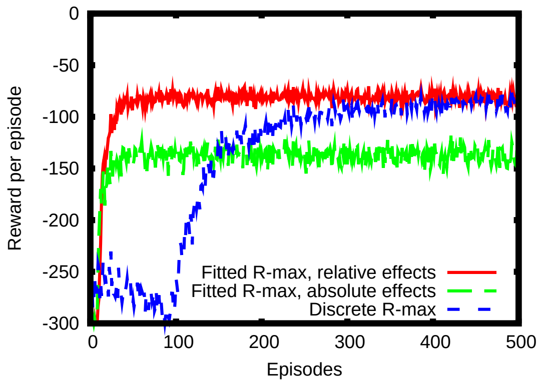
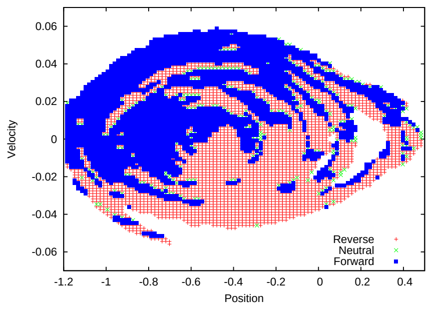


Fig. 8. Learning curves for Mountain Car. Each curve is the average of 50 independent trials.

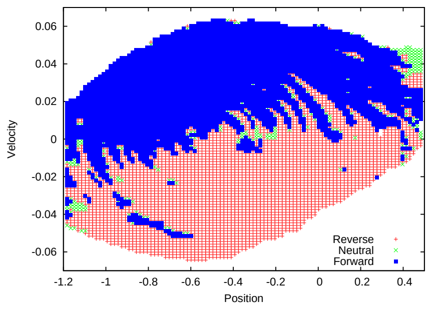
Absolute-transition fitted R-MAXconverges much more quickly than discrete Prioritized Sweeping, but at the expense of converging to suboptimal policies. Further experimentation has shown that decreasing b improves the average quality of the final policy but quickly decreases the learning speed of the algorithm. The standard version of fitted R-MAX uses the more accurate relative transition generalization to preserve fast convergence while achieving near-optimal policies in this domain. For comparison, Figure 9 illustrates typical learned policies for both versions of fitted R-MAX. 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 fitted R-MAX 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 exploration with stable function approximation. Fitted R-MAX extends the data



(a)



(b)

Fig. 9. Mountain-Car policies learned using (a) absolute-transition fitted R-MAXand (b) standard fitted R-MAX. 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.

efficiency of model-based methods to continuous systems, which previously presented the difficulty of representing continuous models. [4] addressed this problem in the deterministic case, also using locally weighted learning from instances. Their application of locally weighted regression estimated the average successor state for each state-action pair; fitted R-MAX approximates the distribution over successor states and thus copes with forms of stochasticity beyond simple noise. They also did not address the issue of exploration in continuous systems. Fitted R-MAX permits the application of intelligent exploration mechanisms originally designed for finite systems. It employs the same mechanism as Prioritized Sweeping [9] and R-MAX [7], 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. Fitted R-MAX 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; fitted R-MAX uses a learned model to generalize both broadly and accurately.

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 one-step dynamics than in the optimal value function. This observation suggests a model-based solution that generalizes first from data to a model. Fitted-model algorithms such as fitted R-MAX apply generalization both to the model and to the value function. They derive a finite representation of the system that both allows efficient planning and intelligent exploration. These attributes allow fitted R-MAX to learn some standard benchmark systems more efficiently than many contemporary RL algorithms.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
2. Watkins, C.: Learning From Delayed Rewards. PhD thesis, University of Cambridge (1989)
3. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. In: Proceedings of the Fifteenth International Conference on Machine Learning. (1998) 260–268
4. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning for control. Artificial Intelligence Review **11** (1997) 75–113
5. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research **4** (2003) 1107–1149
6. Gordon, G.J.: Stable function approximation in dynamic programming. In: Proceedings of the Twelfth International Conference on Machine Learning. (1995)
7. Brafman, R.I., Tennenholtz, M.: R-MAX – a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research **3** (2002) 213–231
8. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc. (1994)
9. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning **13** (1993) 103–130
10. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. (1995)
11. Kekade, S.M.: On the Sample Complexity of Reinforcement Learning. PhD thesis, University College London (2003)
12. Dutech, A., Edmunds, T., Kok, J., Lagoudakis, M., Littman, M., Riedmiller, M., Russell, B., Scherrer, B., Sutton, R., Timmer, S., Vlassis, N., White, A., Whiteson, S.: Reinforcement learning benchmarks and bake-offs II. http://www.cs.rutgers.edu/˜mlittman/topics/nips05-mdp/bakeoffs05.pdf (2005)
13. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Advances in Neural Information Processing Systems 8. (1996)