Model-based Policy Gradient Reinforcement Learning

Xin Wang WANGXI@CS.ORST.EDU
Thomas G. Dietterich TGD@CS.ORST.EDU
Department of Computer Science, Oregon State University, Dearborn Hall 102, Corvallis, OR 97330

Abstract

Policy gradient methods based on REINFORCE are model-free in the sense that they estimate the gradient using only online experiences executing the current stochastic policy. This is extremely wasteful of training data as well as being computationally inefficient. This paper presents a new model-based policy gradient algorithm that uses training experiences much more efficiently. Our approach constructs a series of incomplete models of the MDP, and then applies these models to compute the policy gradient in closed form. The paper describes an algorithm that alternates between pruning (to remove irrelevant parts of the incomplete MDP model), exploration (to gather training data in the relevant parts of the state space), and gradient ascent search. We show experimental results on several benchmark problems including resource-constrained scheduling. The overall feasibility of this approach depends on whether a sufficiently informative partial model can fit into available memory.

1. Introduction

Policy-gradient methods for reinforcement learning work by performing gradient ascent search in a parameterized space of policies to find a policy that optimizes the expected return (at least locally). Obviously this requires computing the gradient of the return with respect to the parameters of the policy. Most policy gradient methods do this by collecting Monte Carlo samples of the current policy executing on the MDP (Williams, 1992; Baxter & Barlett, 2000; Sutton et al., 2000; Konda & Tsitsiklis, 2000; Peshkin et al., 1999). Unfortunately, Monte Carlo estimates tend to have high variance, which means that large numbers of samples are needed to estimate each gradient accurately, and consequently, that learning is very slow. Hence, recent research has focused on ways to reduce the variance of these Monte Carlo estimates.

One general approach to reducing the high variance of Monte Carlo estimates is to use an additive control variate as the reinforcement baseline. Williams (1992) suggests that the average reward baseline could greatly enhance the convergence speed of a policy gradient method. Sutton et al. (2000) and Konda and Tsitsiklis (2000) prove that a special form of value function can be used as a good control variate. Greensmith et al. (2002) discovered that using the average discounted reward or even the true value function as the control variate can lead to sub-optimal solutions. The reason is that parameterized baselines or value function control variates are usually treated as a constant in the computation to obtain the gradient of the policy. Greensmith, et al. show how to compute the optimal constant baseline.

In addition to control variates, several other strategies have been proposed to reduce variance and/or reduce the amount of Monte Carlo sampling needed. Shelton (2001) showed how to estimate the policy gradient by applying importance sampling to stored experiences. Ng et al. (2000) show how to exploit pseudo-random number generators to convert stochastic environment models into deterministic models, which reduces the problem of policy search in an arbitrary MDP to policy search in a deterministic MDP (Kearns et al., 2000). Williams (1992) also suggested that model-based methods could be used to reduce the variance and to speed up the convergence of policy gradient methods.

An alternative to online Monte Carlo gradient estimates is to learn a model of the MDP by recording the agent's interactions with the environment. The model can then be analyzed to compute the gradient. Research pursuing this approach has focused on compact belief-network models of the transition matrix P

and compact models of the reward function R of the MDP. Ng et al. (2000) describe an algorithm that represents the *"time slice distribution"* with a dynamic Bayesian network, and then conducts policy gradient ascent through the network. Boutilier et al. (1995) describe symbolic value iteration and policy iteration algorithms. Guestrin et al. (2001) present approximate policy iteration algorithms that are guaranteed to converge to a good approximation of the optimal policy. Kim et al. (2000) describe an alternative policy representation for Factored MDPs: finite state machine controllers (FSC). Aberdeen and Baxter (2002) describe an algorithm based on an exact FSC model of the world and an algorithm that keeps track of the distribution of the internal FSC-based belief states. Unfortunately, most MDPs are not structured to be factorable. The probability transition matrices of some MDPs are unlikely to correspond to a compact belief network, such as the MDP formulated to learn the search control heuristics in Zhang and Dietterich (1995).

In this paper, we investigate a new policy gradient algorithm that explicitly learns a (partial) tabular model of the transition matrix P and the reward function R of an MDP. This model is solved to compute the value function of the current policy and its expected number of visits to each state. These quantities permit an exact closed-form computation of the policy gradient (for the partial model).

This model-based algorithm does not suffer from the high variance that affects REINFORCE and related online sampling-based methods. Rather, its accuracy is determined by the thoroughness of its exploratory searches. The algorithm consists of a two-phase iterative process that interleaves additional exploration with the gradient ascent search. For large-scale MDPs such as resource-constrained scheduling, it is impossible to store the entire MDP in a tabular form. The overall feasibility of this approach depends on whether a sufficiently informative partial model can fit into available memory. To solve this problem, we introduce effective exploration and pruning heuristics. We demonstrate the effectiveness of the algorithm and the exploration and pruning heuristics on three toy domains and on resource-constrained scheduling problems.

The remainder of this paper is organized as follows: in Section 2, we define the notation that will be used in this paper and introduce the concept of the expected visit function N^π. In Section 3, we describe our algorithm, **MBPG**, its gradient formula, and our exploration and pruning strategy. In Section 4, we present the experimental results of our algorithm in three toy-domains and in the resource-constrained scheduling problems.

2. Preliminaries

Consider a discrete time finite-horizon MDP M: $\langle S, A, S_0, P, R, P_0\rangle$, where S is a finite set of states, A is a finite set of actions, $P(s'|s,a)$ is the probability of making a transition to state s' after executing action a in state s, and $R(s'|s,a)$ is the one-step reward received from executing a in s and moving to s'. A trial begins by choosing a starting state $s_0 \in S_0 \subseteq S$ according to the starting state distribution $P_0(s_0)$. The goal of the agent is to learn a parameterized stochastic policy $\pi(s,a;\theta)$ (the probability of executing action a in state s), $\theta \in \mathcal{R}^n$, that maximizes $J^\pi(\theta)$, the expected total reward received by the agent. In this paper, we assume that all (stochastic) policies over M are proper (i.e., enter a terminal state with non-zero probability).

Given a policy π_θ, we define $V^\pi(s)$ as the value of the state s (the expected cumulative reward received if the agent starts in state s and executes policy π_θ until a terminal state is encountered). Then the expected total reward received by π_θ with respect to the transition model P can be written as

$$J^\pi(\theta) = \sum_{s\in S_0} P_0(s)V^\pi(s). \qquad (1)$$

Let us denote by $N^\pi(s)$ the expected number of visits to state s when executing a trial according to policy π_θ. N^π can be written recursively as the probability $P_0(s)$ of starting in s plus the sum of $N^\pi(s_p)$ of all of s's predecessor states s_p, weighted by the probability of executing the policy π_θ in state s_p and arriving in state s. If the transition probabilities $P(s|s_p,a)$ are known, N^π can be computed by iterating the following formula:

$$\begin{aligned} N^\pi(s;\theta) \quad &:= \quad \sum_{s_p} N^\pi(s_p;\theta) \sum_a \pi(s_p,a;\theta) P(s|s_p,a) \\ &\quad\; + P_0(s). \qquad (2) \end{aligned}$$

This definition is very similar to that of the *influence* function introduced by Munos and Moore (2002), the time slice distribution of Ng et al. (2000), and the internal belief state distribution of Aberdeen and Baxter (2002).

For the dynamic program in Eq 2 to converge, we must assume that for each $\theta \in \mathcal{R}^K$, the corresponding policy $\pi(s,a;\theta)$ has a stationary distribution $\Phi^\pi(\theta)$, which is true for MDPs in which all policies are proper. This is similar to the Assumption 1 of Baxter and Barlett

(2000). Ng et al. (2000) does not explicitly require the assumption to be true, but for infinite-horizon MDPs, their assumption (the limiting distribution of the time slice distribution $\phi_t^\pi(s)$, $\phi_\infty^\pi(s)$, exists) automatically entails it.

3. Algorithm

To find a parameter vector $\theta \in \mathcal{R}^k$ that gives a local maximum of $J(\theta)$, we propose to (partially) explore the MDP and build an (incomplete) model of the states, actions, and transitions. This means that we will learn the transition matrix $P(s|s',a)$ and the reward function $R(s|s',a)$ explicitly. With the model known, the "influence" function N^π can be computed directly from the model by dynamic programming (Eq 2).

Assuming that for each $\theta \in \mathcal{R}^k$ there exists a stationary distribution $\Phi^\pi(s;\theta)$ for π_θ, it can be proved that the dynamic program to compute Eq 2 will converge.[1]

Let $R^\pi(s) = \sum_a \pi(s,a;\theta) \sum_{s'} P(s'|s,a)R(s'|s,a)$ be the expected reward received in state s under policy π. Then $J^\pi(\theta)$ as given by Eq 1 can be calculated by multiplying the expected number of visits to s by the expected reward received on each visit, summed over all states:

$$J^\pi(\theta) = \sum_{s\in S} N^\pi(s;\theta)R^\pi(s). \qquad (3)$$

With $R^\pi(s)$, the Bellman Equations

$$V^\pi(s;\theta) = \sum_a \pi(s,a;\theta) \sum_{s'} P(s'|s,a) [R(s'|s,a) + V^\pi(s';\theta)], \qquad (4)$$

can be rewritten as

$$V^\pi(s;\theta) = R^\pi(s) + \sum_a \pi(s,a;\theta) \sum_{s'} P(s'|s,a)V^\pi(s';\theta)].$$

Let Ψ be the space of all real-valued functions over S, and $\phi, \psi \in \Psi$ be two such functions. Let $\Gamma_{P,\theta} : \Psi \to \Psi$ be a mapping defined such that $\phi = \Gamma_{P,\theta}\psi$ iff

$$\phi(s) = \psi(s) + \sum_a \pi(s,a;\theta) \sum_{s'} P(s'|s,a)\phi(s').$$

Then from Eq 1 and Eq 3, we can obtain the following lemma:

Lemma 1 *For all $\phi(s), \psi(s) \in \Psi$ such that $\phi(s) = \Gamma_{P,\theta}\psi(s)$,*

$$\sum_{s_0 \in S_0} P_0(s_0)\phi(s) = \sum_{s\in S} N^\pi(s)\psi(s).$$

[1]Munos and Moore (2002) give a convergence proof for continuous state MDPs.

3.1. The Gradient

To search in policy space $\pi(s,a;\theta) : \theta \in \mathcal{R}^K$, we need to compute the gradient of Eq 4 with respect to the policy parameters θ.

Theorem 1 *If for each $\theta \in \mathcal{R}^K$, there exists a stationary distribution $\Phi^\pi(s,\theta)$ for a policy $\pi(s,a;\theta)$, $N^\pi(s)$ is given by Eq 2, and the derivatives of $\pi(s,a;\theta)$*

$$\nabla_\theta \pi(s,a;\theta) = \left[\frac{\partial \pi(s,a;\theta)}{\partial \theta_k}\right], k = 1, 2, \cdots, K$$

exist, then the gradient of $J^\pi(\theta)$ can be computed by

$$\nabla_\theta J^\pi(\theta) = \sum_s N^\pi(s;\theta) \sum_a \nabla_\theta \pi(s,a;\theta) \sum_{s'} P(s'|s,a)[R(s'|s,a) + V^\pi(s';\theta)]. \qquad (5)$$

Proof: To simplify the representation, let

$$R(s,a) = \sum_{s'} P(s'|s,a)R(s'|s,a).$$

Then Eq 4 can be rewritten as

$$V^\pi(s) = \sum_a \pi(s,a;\theta) \left[R(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s')\right]$$

By applying the chain rule to V^π, we obtain

$$\frac{\partial V^\pi(s)}{\partial \theta_k} = \sum_a \left[\frac{\partial \pi(s,a;\theta)}{\partial \theta_k}[R(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s')] + \pi(s,a;\theta) \sum_{s'} P(s'|s,a)\frac{\partial V^\pi(s')}{\partial \theta_k}\right].$$

Let $BUV(s,a) = R(s,a) + \sum_{s'} P(s'|s,a)V^\pi(s')$ be the "backed-up value" of executing action a in state s. To simplify the representation, let $G_k(s) = \frac{\partial V^\pi(s)}{\partial \theta_k}$. Then

$$G_k(s) = \sum_a \left(\frac{\partial \pi(s,a;\theta)}{\partial \theta_k} BUV(s,a)\right) + \sum_a \pi(s,a;\theta) \sum_{s'} P(s'|s,a)G(s').$$

Let $F_k(s) = \sum_a \frac{\partial \pi(s,a;\theta)}{\partial \theta_k} BUV(s,a)$. Then $G_k(s)$ can be rewritten as

$$G_k(s) = \Gamma_{P,\theta}F_k(s).$$

From Lemma 1, we have

$$\sum_{s\in S0} P_0(s_0)G_k(s_0) = \sum_s N^\pi(s)F_k(s),$$

which is equivalent to Eq 5. □

This theorem shows that we can compute the gradient in closed form by taking the current policy π_θ, performing value iteration to compute V^π, performing the iteration of Eq 2 to compute N^π, and then visiting every state and evaluating Eq 5.

Table 1. Algorithm MBPG

```
Initialize:
    $\theta_i = 0$, or a random number from $-0.1$ to $0.1$
        $i = 0, 1, \ldots, K$
    $N^\pi(s) = P_0(s)$,
    $V^\pi(s) = 0$,
    $\nabla_\theta \pi(s, a; \theta) = 0$

Repeat until convergence:
    Exploration:
        1. Remove unneeded $(s, a)$ pairs from the model
           $P(s'|s, a)$ and $R(s'|s, a)$
        2. Explore using the current stochastic policy
           $\pi(s, a; \theta)$
        3. Update the model.
    Gradient Ascent:
        3. Compute the expected number of visits to
           each state $N^\pi(s)$ by,
           $N^\pi(s) = \sum_{s_p} N^\pi(s_p) \sum_a \pi(s_p, a; \theta) P(s|s_p, a)$
                  $+ P_0(s)$
        4. Compute $V^\pi(s)$ by value iteration,
           $V^\pi(s) = \sum_a \pi(s, a; \theta) \sum_{s'} P(s'|s, a)$
                  $[R(s'|s, a) + V^\pi(s')]$
        5. Compute $\nabla_\theta \pi(s, a; \theta)$ by
           $\nabla_\theta J^\pi(\theta) = \sum_s N^\pi(s) \sum_a \nabla_\theta \pi(s, a; \theta)$
                  $\sum_{s'} P(s'|s, a)[R(s'|s, a) + V^\pi(s')]$.
        6. Perform a line search in the direction of
           $\nabla_\theta \pi(s, a; \theta)$. This requires repetition of
           steps 4, 5, and 6, to update the gradient.
```

3.2. Details of the Algorithm

A simple, but impractical, application of Theorem 1 would be to first explore the MDP exhaustively to form accurate models of $P(s'|s, a)$ and $R(s'|s, a)$ for all states s and actions a. Then a gradient ascent search could be performed by repeatedly computing the gradient (which requires recomputing V^π and N^π), and modifying θ according to $\theta := \theta + \eta \nabla_\theta J(\theta)$. This may seem pointless, since once we have P and R, we could just apply standard value iteration or policy iteration to compute the optimal policy π^*. However, when we adopt a parameterized representation of the policy, π_θ, our goal is actually no longer to find the optimal policy, but to find the best policy *representable* as a parameterized policy. As the research in value function approximation has shown, this is a very difficult problem, because we no longer can rely on the Bellman equation to converge.

Our strategy instead is to perform a small amount of exploration to learn partial models of P and R. From these, we can compute V^π and N^π and perform a gradient ascent line search to update π_θ (which requires that we update V^π and N^π repeatedly). Then we can remove from our partial model state-action pairs that appear to have no relevance to π_θ and perform additional exploration to add new state-action pairs to the model that are relevant to improving π_θ. Then we can again compute V^π and N^π and perform another gradient ascent line search. This process of adapting the partial model and the parameterized policy continues until a local optimum is reached. We can visualize this as a process of shining a "flashlight beam" on a particular region of the state space, exploring in that region, and then improving π_θ. Then the "flashlight beam" is shifted a bit toward more important parts of the state space, and the process repeats. The empirical question is how large a "flashlight beam" is needed to lead us to a good policy? How much of the state space do we need to remember in our current model? We will describe experimental results for a particularly "narrow" flashlight beam below. Table 1 gives the pseudo-code of our method.

We adopt a line search procedure similar to the GSEARCH algorithm for GPOMDP of Baxter and Barlett (2000), which uses only estimates of the gradient, $\nabla_\theta J^\pi(\theta)$. GSEARCH has problems detecting extreme overshooting of the maximum of $J^\pi(\theta)$. Our algorithm MBPG suffers the overshooting problem even more than GPOMP, because we choose to explore only after each line search. Our model estimates become more and more out-dated as the policy moves farther and farther away in the direction of gradient. To avoid this overshooting problem, we terminate the line search early, before it converges to a local optimum.

In our experiments, we found that, for simple MDPs (such as the three toy domains in 4.1), it is sufficient to explore by following the stochastic policy π_θ. The gradient ascent process converges to the optimal policy quickly. But for large MDPs that have comparatively more expensive simulated actions, we found that this strategy converges too slowly. Table 2 summarizes one of the heuristic exploration strategies we found to be very efficient in our experiments. It is based on the LDS search of Harvey and Ginsberg (1995).

From Eq 3, we can see that MBPG only needs enough samples to correctly estimate the subset of states with high influence (N^π) values under π_θ. For MDPs that have a large state space but a small set of starting states, this set is very small, because $N^\pi(s)$ of a state s decreases exponentially as its distance from S_0 in-

Table 2. Exploration and Pruning Algorithm

```
Let $N$ be the maximum number of exploration steps
Let $N_p$ be the maximum number of paths to consider

set $T = 0$
while ($T < N$)
  1.Identify paths $p_i, i = 1, \dots, N_p$ with the largest
    $N^\pi$ values. Keep the models $P(s'_{i,j}|s_{i,j}, a)$
    and $R(s'_{i,j}|s_{i,j}, a)$ of state $s_{i,j}$ on path $p_i$. Prune
    the rest of the model.
  2.Choose a state-action pair $(s, a)$ to explore from:
    (1) choose a path $p_i$ uniformly at random
    (2) choose a state $s_j$ on $p_i$ uniformly at random
    (3) choose an unexplored action $a'$ uniformly at
        random; if there is no unexplored action, choose
        an action $a$ according to the current policy $\pi_\theta$
  3.From $(s, a)$, explore using $\pi_\theta$ and update the model
    $P$ and $R$. For each action $a$ executed, $T := T + 1$.
```

creases. For these cases, the accuracy of the estimates of $\nabla_\theta J(\theta)$ depends more on the states with dominating $N^\pi(s)$ values. So our exploration strategy needs to cover these states more often to get their $P(s'|s, a)$ and $R(s'|s, a)$ correct. Of course, we still need to cover enough of the "subtree" of the MDP under these states to have accurate estimates of V^π. So to speed up our algorithms for large episodic MDPs, we prune our current model by removing states with low N^π values, and explore only from states that have high N^π values.

4. Experimental Results

For our experiments, we represent the parameterized stochastic policy $\pi(s, a; \theta)$ with the softmax function of a set of potential functions $M(s, a; \theta)$. Let $A(s)$ be the actions in s that have been explored, $\pi(s, a; \theta)$ is computed as

$$\pi(s, a; \theta) = \frac{e^{M(s,a;\theta(a))}}{\sum_{u \in A(s)} e^{M(s,u;\theta(u))}}$$

where $\theta(a)$ is the portion of θ for action a.

We have employed four representations for the potential functions: (a) a tabular representation with one value $\theta(s, a)$ for each (s, a) pair, (b) a linear combination of state-action features $x_i(s, a)$, $i = 1, \dots, I$: $M(s, a; \theta) = \sum_{i=1}^{I} \theta_i x_i(s, a) + \theta_0$, (c) a neural network (we used networks with a single hidden layer of sigmoid units, and an output layer of linear units), and (d) a CMAC (Sutton & Barto, 1998).

4.1. Three Toy Domains

First, to test our algorithm, we applied it to three toy domains: the 4x3 world (Russell & Norvig, 1995), the mountain car problem (Sutton & Barto, 1998), and the 100x100 maze (Dietterich & Wang, 2002).

4.1.1. 4x3 World

The 4x3 world has 9 non-terminal states, one good goal state, and one bad goal state. The agent can move in any of four directions: *east*, *south*, *west* or *north*. Each move has an equal probability of 10% to turn into the right or the left cell of its original destination. For the cells on the boundary, an action that moves the agent into the wall will just keep the agent in the same cell. Each move has a negative reward of -0.04. The starting state is chosen uniformly from the non-terminal states. If the agent ends up in the good goal state, it gets a reward of 0.96. If it ends up in the bad goal state, it gets a negative reward of -1.04.

On this problem, we experimented with both the table and the linear policy potential function (for the 4x3 problem, the optimal policy is representable by a linear potential function). For both representations, **MBPG** converges to the optimal policy. With a table, **MBPG** converges in one line search after 1000 steps of exploration. With a randomly initialized linear potential function, the algorithm takes 9 to 13 line searches to converge to the optimal policy (with 1000 steps of exploration per line search). We also implemented GPOMDP for the 4x3 world. With the tabular potential functions, GPOMDP takes from 5000 to 150000 steps (67000 steps on the average) to converge to the optimal policy.

4.1.2. Mountain Car

For the mountain car problem, we experimented with three different policy potential function forms: tables, CMACs, and neural networks. In all three cases, **MBPG** converged to the optimal policy. For the table representation, we used a 100x100 discretization of the state space. We also experimented with a 10-tile CMAC with 8 cells in each dimension (as in (Sutton & Barto, 1998)). For both CMAC and table representations of the policy, the algorithm learns at about the same speed: it takes **MBPG** 3-8 million exploration steps to converge to the optimal policy.

4.1.3. 100x100 maze

One of the open research questions in reinforcement learning is how to transfer a policy learned from one or a few problem instances to other similar ones. For example, in the job-shop scheduling work of Zhang

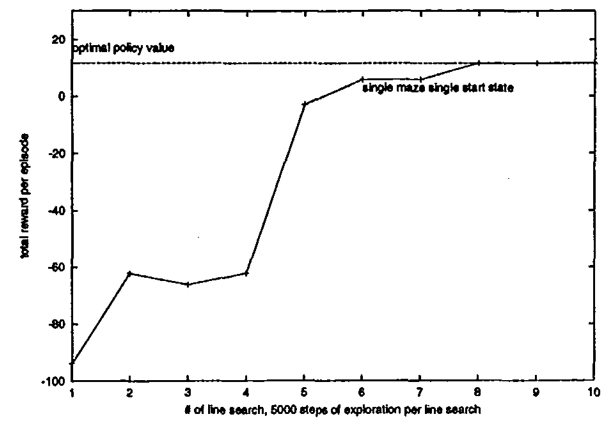

Figure 1. Performance on single 100*100 maze and one single start state.

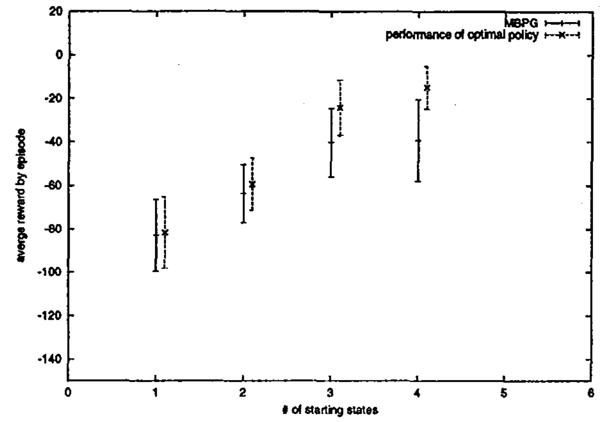

Figure 2. Performance on single 100x100 maze but multiple starting states

and Dietterich (1995), the goal is to learn a policy that works well across a whole distribution of possible scheduling problem instances by training on only a sample of instances from this distribution. Such a policy is typically not optimal for any particular instance—it represents a "compromise" across all of the instances. It is not obvious how to solve this compromise problem with value function methods, because the value functions may have different values on different problem instances, but policy gradient methods provide an easy solution: We can average the policy gradient across the given problem instances, and then apply gradient ascent search.

To test this approach, we generated multiple random 100x100 mazes according to the specifications in Dietterich and Wang (2002). These mazes contain penalties drawn at random according to an underlying distribution, so we know that the optimal "compromise" policy is the optimal policy for the underlying penalty distribution, and yet the optimal policy for each individual maze can be radically different. With these mazes, we can test generalization ability in two ways. First, we can train using only a small number of starting states, and then see how well the learned policy works from other starting states. Second, we can train using only a small number of mazes, and see how well the learned policy works on new mazes generated according to the same penalty distribution.

On a single 100x100 maze with one single starting state, as shown in Fig. 1, **MBPG** converges to the (single-maze) optimal policy within 40000 exploration steps. Fig. 2 shows the performance of **MBPG** trained on a single maze with *multiple* starting states. The performance of the policy is evaluated by executing the policy on a separate set of 20 randomly generated starting states. The error bars show the standard deviation over 5 different runs of the algorithm. The policy gradient method is able to generalize to new states quite well. For comparison, we plot the optimal policy computed from the learned model. There are no statistically significant differences.

Next, we trained **MBPG** on multiple mazes with four randomly-selected start states per maze. We evaluated the learned policy on five additional mazes with four start states each. On this toy problem, it is possible to compute an "averaged" model by pooling the results of exploring the training mazes, and the optimal policy of this merged model will be the optimal compromise policy (with enough exploration). Fig. 3 compares the performance of policy gradient (computed by averaging the gradients computed on the individual training problems) with the policy computed from the merged model. We see that **MBPG** is giving better results than the averaged model when trained on 3 or more mazes.

4.2. Job Shop Scheduling Problems

The job shop scheduling problems of Zhang and Dietterich (1995) are based on the NASA Space Shuttle Payload Processing domain. These problems are more difficult than classical job shop scheduling problems because the resources (fork lifts, test stations, etc.) are grouped into resource pools. When a task is assigned resources of a particular type, it must draw all of those resources from a single pool. We have experimented with the ART-1 benchmark data set. Each ART-1 instance consists of 2-6 jobs. Each job consists of 8-15 partially ordered tasks (about 60% of all possible constraints are asserted). Each task randomly chooses to require 1 to 2 types of resources. Each resource has 2

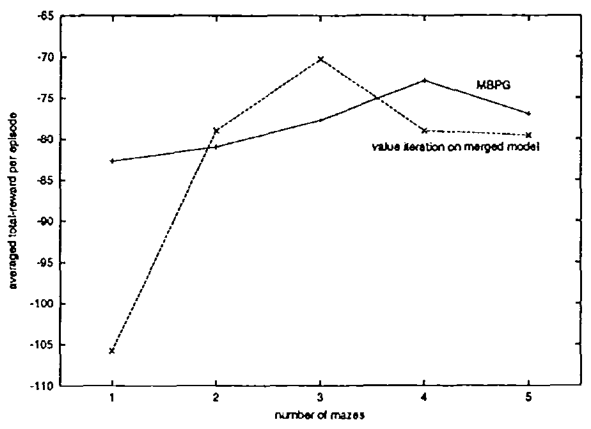

Figure 3. Results on training on multiple 100x100 mazes and multiple start states.

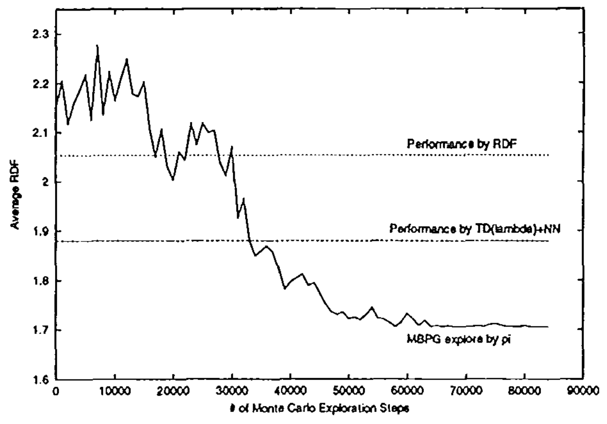

Figure 4. MBPG on job shop scheduling problem ART-1:trn00.

pools (one with capacity 6, the other of capacity 8). The scheduling process can be modelled as an MDP. Each state s of the MDP is a partial schedule. Each action is a possible repair to a task that has violated constraints. The goal is to find the shortest violation free schedule. For a repair action leading to a goal state, the reward is the RDF (Resource Dilation Factor) of the schedule; otherwise, the reward is -0.001. A smaller RDF value means that the algorithm has found a better final violation free schedule.

The search depths from a starting state to a goal state range from 40 to more than 100 repair actions. So the resulting MDP has a very large state space, but it is sparse (out-degree around 4 per state) and mostly tree-structured. A challenging aspect of this problem is that the set of actions depends on the set of available repairs in a state, so actions do not have a fixed identity, which makes it difficult to define features x_i of actions. Fortunately, the actions are deterministic (at least if we ignore state-aliasing introduced by the features). Hence, we can represent each action by the state s' that it results in.

Fig. 4 shows the behavior of **MBPG** on a single job shop scheduling problem (ART-1:TRN00) exploring using the current policy π_θ with no model pruning. For comparison, we show the performance of a greedy RDF-related heuristic and the performance of a neural network trained by $TD(\lambda)$ (Zhang & Dietterich, 1995) on a set of 20 training problems (including ART-1:TRN00). We see that **MBPG** performs much better. However, the comparison with the neural networks is not entirely fair, because the neural network was trained on 20 problems, so its policy represents a compromise across those problems.

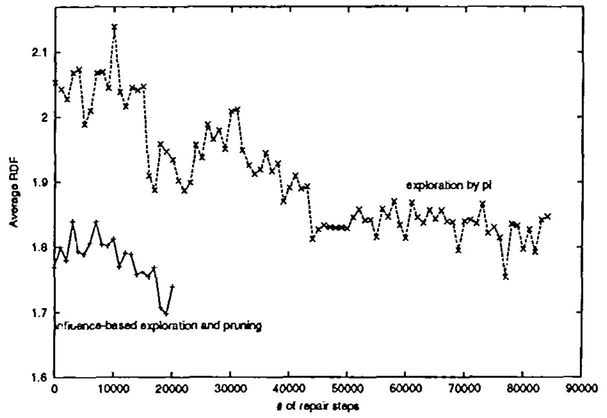

Figure 5. MBPG performances on a 20 problem test set, exploring with LDS strategy vs. exploring according to the policy.

Fig. 5 compares the performance of **MBPG** with two different exploration strategies: (a) exploration using π_θ with no pruning and (b) exploration using the N^π method of Table 2. The N^π method converges in 20000 steps compared to more than 75000 for π_θ. Fig. 6 shows that, with the exception of 2 instances, training on more problem instances leads **MBPG** to converge to better results in fewer steps.

5. Conclusion and Future Work

This paper has presented a model-based policy gradient algorithm, **MBPG**, that relies on building and solving a series of partial models of the MDP to compute the policy gradient in closed form. This gives a policy gradient search that needs many fewer training examples than methods, such as GPOMDP and RE-

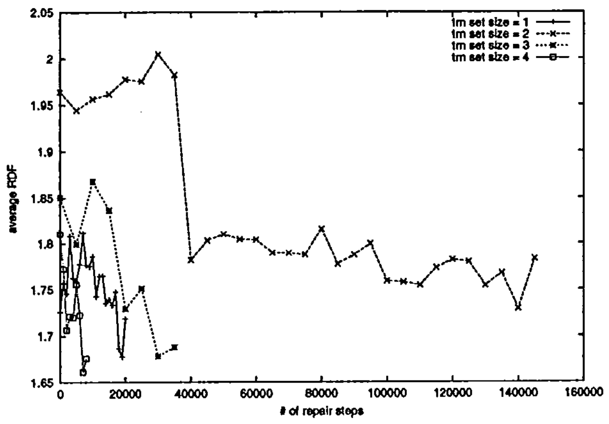

Figure 6. **MBPG** performance of training on multiple problems (influence-based exploration).

INFORCE, that estimate the gradient through Monte Carlo trials. We have seen experimentally that even on the trivial 4x3 maze problem, **MBPG** learns much faster than GPOMDP (the most effective Monte Carlo policy gradient method).

The feasibility of **MBPG** depends on whether useful gradients can be computed from relatively small partial models of the MDP and whether good strategies can be found for pruning and exploration to construct those models. The paper introduced one such pruning and exploration method that exploits the expected number of visits N^{π} computed as part of the gradient calculation to make its decisions. We showed experimentally that this method gives large speedups in our primary application domain—resource-constrained scheduling.

Future work will focus on understanding the impact of restricting the size of the partial model and on developing theoretically better-motivated pruning and exploration heuristics.

Acknowledgements

The authors gratefully acknowledge the support of the NSF under grants IIS-0083292 and ITR-0085836. We also thank Thomas Amoth and Prasad Tadepalli for several helpful suggestions.

References

Aberdeen, D., & Baxter, J. (2002). Scalable internal-state policy-gradient methods for POMDPs. *ICML-2002* (pp. 3–10). Morgan Kaufmann.

Baxter, J., & Barlett, P. (2000). Reinforcement learning in POMDPs via direct gradient ascent. *ICML-2000* (pp. 41–48). Cambridge, MA: The MIT Press.

Boutilier, C., Dearden, R., & Goldszmidt, M. (1995). Exploiting structure in policy construction. *IJCAI-1995* (pp. 1104–1111). San Francisco: Morgan Kaufmann.

Dietterich, T. G., & Wang, X. (2002). Batch value function approximation via support vectors. *NIPS-2001* (pp. 1491–1498). Cambridge, MA: The MIT Press.

Greensmith, E., Bartlett, P., & Baxter, J. (2002). Variance reduction techniques for gradient estimates in reinforcement learning. *NIPS-2001* (pp. 1507–1514). Cambridge, MA: The MIT Press.

Guestrin, C., Koller, D., & Parr, R. (2001). Max-norm projections for factored MDPs. *IJCAI-2001* (pp. 673–682).

Harvey, W. D., & Ginsberg, M. L. (1995). Limited discrepancy search. *IJCAI-95* (pp. 607–615). Montréal, Québec, Canada: Morgan Kaufmann, 1995.

Kearns, M., Mansour, Y., & Ng, A. Y. (2000). Approximate learning in large POMDPs via reusable trajectories. *NIPS-1999* (pp. 1001–1007). Cambridge, MA: The MIT Press.

Kim, K.-E., Dean, T., & Meuleau, N. (2000). Approximates solutions to factored markov decision processes via greedy search in the space of finite state controllers. *Artificial Intelligence Planning Systems* (pp. 323–330).

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-critic algorithms. *NIPS-1999.* Cambridge, MA: The MIT Press.

Munos, R., & Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning, 49,* 291–323.

Ng, A. Y., Parr, R., & Koller, D. (2000). Policy search via density estimation. *NIPS-1999.* Cambridge, MA: The MIT Press.

Peshkin, L., Meuleau, N., & Kaelbling, L. P. (1999). Learning policies with external memory. *ICML-1999* (pp. 307–314). Cambridge, MA: The MIT Press.

Russell, S., & Norvig, P. (1995). *Artificial intelligence, a modern approach.* Prentice Hall.

Shelton, C. R. (2001). Policy improvement for POMDPs using normalized importance sampling. *UAI-2001* (pp. 496–503).

Sutton, R., & Barto, A. (1998). *Reinforcement learning: an introduction.* Cambridge, MA: MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *NIPS-1999* (pp. 1057–1063). Cambridge, MA: The MIT Press.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8,* 229–256.

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. *IJCAI-1995* (pp. 1114–1120).