Random Sampling of States in Dynamic Programming

Christopher G. Atkeson and Benjamin J. Stephens

***Abstract*—We combine three threads of research on approximate dynamic programming: sparse random sampling of states, value function and policy approximation using local models, and using local trajectory optimizers to globally optimize a policy and associated value function. Our focus is on finding steady-state policies for deterministic time-invariant discrete time control problems with continuous states and actions often found in robotics. In this paper, we describe our approach and provide initial results on several simulated robotics problems.**

***Index Terms*—Dynamic programming, optimal control, random sampling.**

I. Introduction

DYNAMIC programming provides a way to find globally optimal control laws (policies) $\mathbf{u} = \mathbf{u}(\mathbf{x})$, which give the appropriate action $\mathbf{u}$ for any state $\mathbf{x}$ [1], [2]. Dynamic programming takes as input a one-step cost (a.k.a. "reward" or "loss") function and the dynamics of the problem to be optimized. This paper focuses on the offline planning of nonlinear control laws for control problems with continuous states and actions, deterministic time-invariant discrete time dynamics $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$, and a time-invariant one-step cost function $L(\mathbf{x}, \mathbf{u})$. We are focusing on steady-state policies and, thus, an infinite time horizon. We assume that we know the dynamics and one-step cost function and have accurate state estimates. Future work will address simultaneously learning a dynamic model, finding a robust policy, and performing state estimation with an erroneous partially learned model [3]–[5].

One approach to dynamic programming is to approximate the value function $V(\mathbf{x})$ (the optimal total future cost from each state $V(\mathbf{x}) = \min_{\mathbf{u}_k} \sum_{k=0}^{\infty} L(\mathbf{x}_k, \mathbf{u}_k)$) by repeatedly solving the Bellman equation $V(\mathbf{x}) = \min_{\mathbf{u}}(L(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u})))$ at sampled states $\mathbf{x}$ until the value function estimates have converged to globally optimal values [1], [2]. We explore the approximation of the value function and policy using many local models rather than using tables or global parametric models.

A. Example Problem

We use one-link pendulum swingup as an example problem to provide the reader with a visualizable example of a value function and policy. In one-link pendulum swingup, a motor at the base of the pendulum swings a rigid arm from the downward stable equilibrium to the upright unstable equilibrium and balances the arm there (Fig. 1). What makes this challenging is that the one-step cost function penalizes the amount of torque used and the deviation of the current position from the goal. The controller must try to minimize the total cost of the trajectory. The one-step cost function for this example is a weighted sum of the squared position errors (θ: difference between current angle and the goal angle) and the squared torques τ : $L(\mathbf{x}, \mathbf{u}) = 0.1\theta^2 T + \tau^2 T$, where 0.1 weights the position error relative to the torque penalty and T is the time step of the simulation (0.01 s). There are no costs associated with the joint velocity. The uniform density link has a mass of 1 kg, length of 1 m, and width of 0.1 m. Fig. 2 shows the value function and policy.

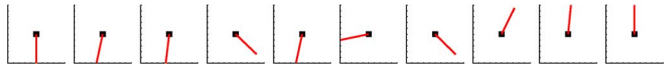

Fig. 1. Configurations from the simulated one-link pendulum swingup optimal trajectory every half second and at the end of the trajectory.

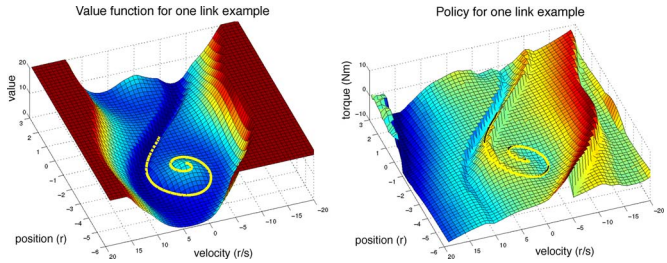

Fig. 2. Value function and policy for a one-link pendulum swingup. The optimal trajectory is shown as a yellow line in the value function plot and as a black line with a yellow border in the policy plot. The value function is cut off above 20 so we can see the details of the part of the value function that determines the optimal trajectory. The goal is at the state (0, 0).

One important thread of our research on approximate dynamic programming is developing value function representations that adapt to the problem being solved and extend the range of problems that can be solved with a reasonable amount of memory and time. Random sampling of states has been proposed by a number of researchers. In our case, we add new randomly selected states as we solve the problem, allowing the "grid" that results to reflect the local complexity of the value function as we generate it.

Another important thread in this paper is developing ways for grids or random samples to be as sparse as possible. One technique that we apply here is to represent full trajectories from each sampled state to the goal and to refine each trajectory using local trajectory optimization [6]. One key aspect of the local trajectory optimizer that we use is that it provides a local quadratic model of the value function and a local linear model of the policy at the sampled state. These local models help our function approximators handle sparsely sampled states. To obtain globally optimal solutions, we incorporate the exchange of information between nonneighboring sampled

Manuscript received September 4, 2007; revised March 3, 2008. This work was supported in part by the DARPA Learning Locomotion Program and in part by the National Science Foundation under Grants CNS-0224419, DGE-0333420, ECS-0325383, and EEC-0540865. This paper was recommended by Guest Editor D. Liu.

C. G. Atkeson is with the Robotics Institute and the Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

B. J. Stephens is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2008.926610

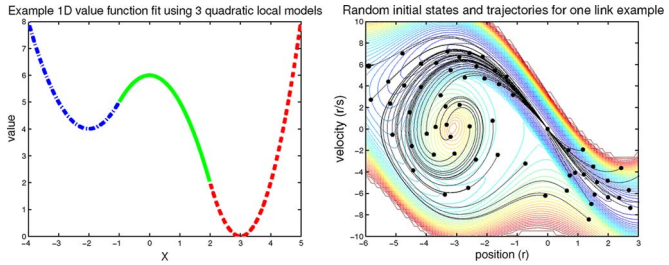

Fig. 3. (Left) example of a local approximation of a 1-D value function using three quadratic models. (Right) (dots) random states used to plan one-link swingup superimposed on a contour map of the value function. (Black lines) optimized trajectories are shown starting from the random states.

states. Fig. 3 (left) shows how local quadratic models can be used to represent a 1-D value function. Fig. 3 (right) shows a randomly generated set of 2-D states superimposed on a contour plot of the value function for one-link swingup and the optimized trajectories used to generate 2-D locally quadratic value function models.

B. On What Problems Will the Proposed Approach Work Well?

We believe that our approach can discover underlying simplicity in many typical problems. An example of a problem that appears complex but is actually simple is a problem with linear dynamics and a quadratic one-step cost function. Dynamic programming can be done for such linear quadratic regulator (LQR) problems even with hundreds of dimensions, and it is not necessary to build a grid of states [7]. The cost of representing the value function is quadratic in the dimensionality of the state. The cost of performing a "sweep" or update of the value function is at most cubic in the state dimensionality. Continuous states and actions are easy to handle. Perhaps, many problems, such as the examples in this paper, have simplifying characteristics similar to LQR problems. For example, problems that are only "slightly" nonlinear and have a locally quadratic cost function may be solvable with quite sparse representations. One goal of this paper is to develop methods that do not immediately build a hugely expensive representation if it is not necessary and attempt to harness simple and inexpensive parallel local planning to solve complex planning problems. Another goal of this paper is to develop methods that can take advantage of situations where only a small amount of global interaction is necessary to enable local planners capable of solving local problems to find globally optimal solutions.

II. Related Work

A. Random State Selection

Random grids and random sampling are well known in numerical integration [8], finite element methods, and partial differential equations [9]. Rust applied random sampling of states to dynamic programming [10]–[13]. He showed that random sampling of states can avoid the curse of dimensionality for stochastic dynamic programming problems with a finite set of discrete actions. In reinforcement learning, random sampling of states is sometimes used to provide training data for function approximation of the value function (e.g., [14]–[16]). Reinforcement learning also uses random exploration for several purposes [17]. In the field of partially observable Markov decision processes, there has been some work on randomly sampling belief states and also using local models of the value function and its first derivative at each randomly sampled belief state (e.g., [18]–[23]). In robot planning for obstacle avoidance, random sampling of states is now quite popular [24]. An alternative to random sampling of states is irregular or adaptive grids [25], but in our experience, they have not performed as well on the example problems as random state approaches.

B. Trajectories and Trajectory Libraries

In our approach, we use trajectories to provide a more accurate estimate of the value of a state. Larson suggested the evaluation of the value in the Bellman equation after several time steps rather than a single step [26]. In reinforcement learning, "rollout" or simulated trajectories are often used to provide training data for approximating value functions [12], [14] as well as evaluating expectations in stochastic dynamic programming. Murray *et al.* used trajectories to provide estimates of the values of a set of initial states [27]. A number of efforts have been made to use collections of trajectories to represent policies [6], [28]–[34]. Schierman *et al.* [30] created sets of locally optimized trajectories to handle changes to the system dynamics. In [6] and [29], information transfer between stored trajectories was used to form sets of globally optimized trajectories for control.

C. Local Models

Werbos proposed the use of local quadratic models of the value function [35]. The use of trajectories and a second-order gradient-based trajectory optimization procedure such as differential dynamic programming (DDP) allows us to use Taylor series-like local models of the value function and policy [36], [37]. Similar trajectory optimization approaches could have been used [38], including robust trajectory optimization approaches [39]–[41]. An alternative to local value function and policy models are global parametric models, for example, [14], [17], and [42]. A difficult problem is choosing a set of basis functions or features for a global representation. Usually, this has to be done by hand. An advantage of local models is that the choice of basis functions or features is not as important. Random sampling of states can also play an important role in training global parametric models. However, this paper focuses on the use of local models.

Parts of this paper have appeared at conferences [43], [44].

III. Combining Random State Sampling With Local Models and Local Optimization

We first present an overview of our approach. We store local quadratic models of the value function and local linear models of the policy at sampled states. The local models are created as a byproduct of our trajectory optimization process. A global approximation of the value function and policy is formed by using the local model for the sampled state closest to the query state. New candidate states are randomly sampled, and a number of acceptance criteria are used to reject redundant states or states whose value estimate is likely to be incorrect. Information is propagated between sampled states to ensure that the Bellman equation is satisfied everywhere and the value function and policy are globally optimal.

A. Local Models of the Value Function and Policy

We need to represent value functions as sparsely as possible. We propose a hybrid tabular and parametric approach: Parametric local models of the value function and policy are represented at sampled locations. This representation is similar to using many Taylor series approximations of a function at different points. At each sampled state $\mathbf{x}^p$, the local quadratic model for the value function is

$$V^p(\mathbf{x}) \approx V_0^p + \mathbf{V}_x^p \hat{\mathbf{x}} + \frac{1}{2}\hat{\mathbf{x}}^{\mathrm{T}} \mathbf{V}_{xx}^p \hat{\mathbf{x}} \tag{1}$$

where $\hat{\mathbf{x}} = \mathbf{x} - \mathbf{x}^p$ is the vector from the sampled state $\mathbf{x}^p$, V_0^p is the constant term of the local model, $\mathbf{V}_x^p$ is the first derivative of the local model (and the value function) at $\mathbf{x}^p$, and $\mathbf{V}_{xx}^p$ is the second derivative of the local model (and the value function) at $\mathbf{x}^p$. The local linear model for the policy is

$$\mathbf{u}^p(\mathbf{x}) = \mathbf{u}_0^p - \mathbf{K}^p \hat{\mathbf{x}} \tag{2}$$

where $\mathbf{u}_0^p$ is the constant term of the local policy and $\mathbf{K}^p$ is the first derivative of the local policy and also the gain matrix for a local linear controller. V_0, $\mathbf{V}_x$, $\mathbf{V}_{xx}$, and $\mathbf{K}$ are stored with each sampled state.

B. Creating the Local Model

These local models of the value function can be created by using DDP [6], [29], [36], [37]. This local trajectory optimization process is similar to LQR design in that a local model of the value function is produced. In DDP, value function and policy models are produced at each point along a trajectory. Suppose, at a point $(\mathbf{x}^i, \mathbf{u}^i)$, we have the following: 1) a local second-order Taylor series approximation of the optimal value function: $V^i(\mathbf{x}) \approx V_0^i + \mathbf{V}_x^i \hat{\mathbf{x}} + (1/2)\hat{\mathbf{x}}^{\mathrm{T}} \mathbf{V}_{xx}^i \hat{\mathbf{x}}$ where $\hat{\mathbf{x}} = \mathbf{x} - \mathbf{x}^i$; 2) a local second-order Taylor series approximation of the robot dynamics, which can be learned by using local models of the dynamics ($\mathbf{f}_x^i$ and $\mathbf{f}_u^i$ correspond to $\mathbf{A}$ and $\mathbf{B}$, respectively, of the linear plant model used in LQR design): $\mathbf{f}^i(\mathbf{x}, \mathbf{u}) \approx \mathbf{f}_0^i + \mathbf{f}_x^i \hat{\mathbf{x}} + \mathbf{f}_u^i \hat{\mathbf{u}} + (1/2)\hat{\mathbf{x}}^{\mathrm{T}} \mathbf{f}_{xx}^i \hat{\mathbf{x}} \; + \hat{\mathbf{x}}^{\mathrm{T}} \mathbf{f}_{xu}^i \hat{\mathbf{u}} + (1/2)\hat{\mathbf{u}}^{\mathrm{T}} \mathbf{f}_{uu}^i \hat{\mathbf{u}}$ where $\hat{\mathbf{u}} = \mathbf{u} - \mathbf{u}^i$; and 3) a local second-order Taylor series approximation of the one-step cost, which is often known analytically for human specified criteria ($\mathbf{L}_{xx}$ and $\mathbf{L}_{uu}$ correspond to $\mathbf{Q}$ and $\mathbf{R}$, respectively, of LQR design): $L^i(\mathbf{x}, \mathbf{u}) \approx L_0^i + \mathbf{L}_x^i \hat{\mathbf{x}} + \mathbf{L}_u^i \hat{\mathbf{u}} + (1/2)\hat{\mathbf{x}}^{\mathrm{T}} \mathbf{L}_{xx}^i \hat{\mathbf{x}} + \hat{\mathbf{x}}^{\mathrm{T}} \mathbf{L}_{xu}^i \hat{\mathbf{u}} + (1/2)\hat{\mathbf{u}}^{\mathrm{T}} \mathbf{L}_{uu}^i \hat{\mathbf{u}}$.

Given a trajectory, one can integrate the value function and its first and second spatial derivatives backward in time to compute an improved value function and policy. We utilize the "Q function" notation from reinforcement learning: $Q(\mathbf{x}, \mathbf{u}) = L(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u}))$. The backward sweep takes the following form (in discrete time):

$$\mathbf{Q}_x^i = \mathbf{L}_x^i + \mathbf{V}_x^i \mathbf{f}_x^i; \quad \mathbf{Q}_u^i = \mathbf{L}_u^i + \mathbf{V}_x^i \mathbf{f}_u^i \tag{3}$$

$$\mathbf{Q}_{xx}^i = \mathbf{L}_{xx}^i + \mathbf{V}_x^i \mathbf{f}_{xx}^i + \left(\mathbf{f}_x^i\right)^{\mathrm{T}} \mathbf{V}_{xx}^i \mathbf{f}_x^i \tag{4}$$

$$\mathbf{Q}_{ux}^i = \mathbf{L}_{ux}^i + \mathbf{V}_x^i \mathbf{f}_{ux}^i + \left(\mathbf{f}_u^i\right)^{\mathrm{T}} \mathbf{V}_{xx}^i \mathbf{f}_x^i \tag{5}$$

$$\mathbf{Q}_{uu}^i = \mathbf{L}_{uu}^i + \mathbf{V}_x^i \mathbf{f}_{uu}^i + \left(\mathbf{f}_u^i\right)^{\mathrm{T}} \mathbf{V}_{xx}^i \mathbf{f}_u^i \tag{6}$$

$$\Delta \mathbf{u}^i = \left(\mathbf{Q}_{uu}^i\right)^{-1} \mathbf{Q}_u^i; \quad \mathbf{K}^i = \left(\mathbf{Q}_{uu}^i\right)^{-1} \mathbf{Q}_{ux}^i \tag{7}$$

$$\mathbf{V}_x^{i-1} = \mathbf{Q}_x^i - \mathbf{Q}_u^i \mathbf{K}^i; \quad \mathbf{V}_{xx}^{i-1} = \mathbf{Q}_{xx}^i - \mathbf{Q}_{xu}^i \mathbf{K}^i \tag{8}$$

where subscripts indicate derivatives and superscripts indicate the trajectory index. After the backward sweep, forward integration can be used to update the trajectory itself: $\mathbf{u}_{\mathrm{new}}^i = \mathbf{u}^i - \Delta \mathbf{u}^i - \mathbf{K}^i(\mathbf{x}_{\mathrm{new}}^i - \mathbf{x}^i)$. We note that the cost of this approach grows at most cubically rather than exponentially with respect to the dimensionality of the state.

C. Utilizing the Local Models

For the purpose of explaining our algorithm, let us assume that we already have a set of sampled states; each of which has a local model of the value function and the policy. How should we use these multiple local models? The simplest approach is to just use the predictions of the nearest sampled state, which is what we currently do. We use a kd-tree to efficiently find nearest neighbors, but there are many other approaches that will find nearby sampled states efficiently [45]. In the future, we will investigate by using other methods to combine local model predictions from nearby sampled states: distance weighted averaging (kernel regression), linear locally weighted regression, and quadratic locally weighted regression for value functions [45].

D. Creating New Random States

For tasks with a goal state, we initialize the set of sampled states by storing the goal state itself. We have explored a number of distributions to select additional states from the following: uniform within bounds on the states; Gaussian with the mean at the goal; sampling near existing states; and sampling from an underlying low resolution regular grid. The uniform approach is a useful default approach, which we use in the swing-up examples, the Gaussian approach provides a simple way to tune the distribution, sampling near existing states provides a way to efficiently sample while growing the solved region in high dimensions, and sampling from an underlying low resolution grid seems to perform well when only a small number of sampled states are used (similar to using low dispersion sequences [10], [24]). A key point of our approach is that we do not generate the random states in advance but instead select them as the algorithm progresses. This allows us to apply an acceptance criteria to candidate states, which we describe in the next paragraph. We have also explored the changing of the distribution we generate candidate states from as the algorithm progresses, for example, using a mixture of Gaussians with the Gaussians centered on existing sampled states. Another reasonable hybrid approach would be to initially sample from a grid and then bias more general sampling to regions of higher value function approximation error.

E. Acceptance Criteria for Candidate States

We have several criteria which are applied in sequence to accept or reject states to be permanently stored [1)–4) hereafter]. In the future, we will explore "forgetting" or removing stored states, but at this point, we apply all memory control techniques at the storage event. To focus the search and limit the volume considered, a value limit is maintained (V_{limit}). This limit is increased to consider a growing volume of state space (Fig. 4). The approximated value function is used to predict the value of the candidate state.

1) If the prediction is above V_{limit}, the candidate state is rejected. Otherwise, a trajectory is created from the candidate state using the current approximated policy and then locally optimized.

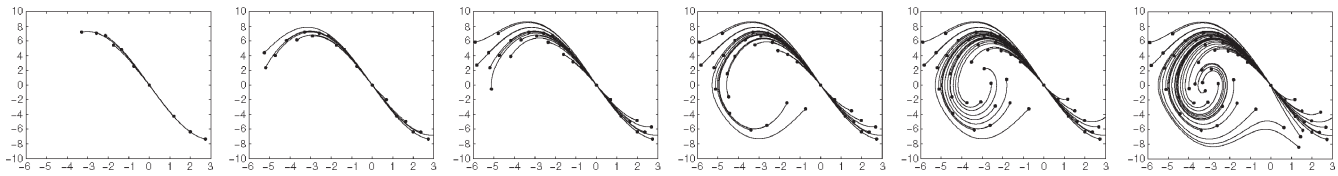

Fig. 4. Sampled states and trajectories for the one-link swingup problem after 10, 20, 30, 40, 50, and 60 states are stored. These figures correspond to Fig. 3 (right), with position on the x-axis and velocity on the y-axis.

2) If the value of that trajectory is above V_{limit}, the candidate state is rejected.
3) If the value of the trajectory is within 10% of the predicted value, the candidate state is rejected. We refer to this as our "surprise" criteria in that only surprises are stored.
4) For problems with a goal state, if the trajectory does not reach the goal, the candidate state is rejected.

Other criteria such as an A*-like criteria (cost-to-go $(\mathbf{x})$ + cost-from-start $(\mathbf{x}) >$ threshold) can be used to reject candidate states [24]. All of the thresholds mentioned can be changed as the algorithm progresses. For example, V_{limit} is gradually increased during the solution process to increase the volume considered by the algorithm. In future work, we will explore starting with a larger surprise threshold and decreasing this threshold with time to further reduce the number of samples accepted and stored while improving convergence. It is possible to take the distance to the nearest sampled state into account in the acceptance criteria for new samples. The common approach of accepting states beyond a distance threshold enforces a minimum resolution and leads to a potentially severe curse of dimensionality effects. Rejecting states that are too close to existing states will increase the error in representing the value function but may be a way for preventing too many samples near complex regions of the value functions that have little practical effect. For example, we often do not need much accuracy in representing the value function near policy discontinuities where the value function has discontinuities in its spatial derivative and "creases." In these areas, the trajectories typically move away from the discontinuities, and the details of the value function have little effect.

F. Creating a Trajectory From a State

We create a trajectory from a candidate state or refine a trajectory from a sampled state in the same way. The first step is to use the current approximated policy until the goal or a time limit is reached. In the current implementation, this involves finding the sampled state nearest to the current state in the trajectory and using its locally linear policy to compute the action on each time step. The second step is to locally optimize the trajectory. We use DDP in the current implementation [6], [29], [36], [37].

G. Combining Parallel Greedy Local Optimizers to Perform Global Optimization

As currently described, the algorithm finds a locally optimal policy but not necessarily a globally optimal policy. For example, the sampled states could be divided into two sets of nearest neighbors. One set could have a suboptimal policy but have no interaction with the other set of states that had a globally optimal policy because no nearest neighbor relations joined the two sets. We expect the locally optimal policies to be fairly good because we 1) gradually increase the solved volume (Fig. 4) and 2) use local optimizers. Given local optimization of actions, gradually increasing the solved volume will result in a globally optimal policy if the boundary of this volume never touches a nonadjacent section of itself. Figs. 2 and 3 show the creases in the value function (discontinuities in the spatial derivative) and corresponding discontinuities in the policy that typically result when the constant cost contour touches a nonadjacent section of itself as V_{limit} is increased.

By enforcing the consistency of the local value function models across all nearest neighbor pairs, we can create a globally consistent value function estimate. Consistency means that any state's local model correctly predicts values of nearby states. If the value function estimate is consistent everywhere, the Bellman equation is solved and we have a globally optimal policy. We can enforce consistency of nearest neighbor value functions by the following: 1) using the policy of one state of a pair to reoptimize the trajectory of the other state of the pair and vice versa and 2) adding more sampled states in between nearest neighbors that continue to disagree [6]. This approach is similar to using the method of characteristics to solve partial differential equations [46] and finding value functions for games [47]–[49].

In theory, the approach we have described will produce a globally optimal policy if it has infinite resolution and all the sampled states form a densely connected set in terms of nearest neighbor relations [6]. The proposed approach becomes similar to a standard dynamic programming approach as the resolution becomes infinite and thus inherits the convergence properties of grid-based dynamic programming [1], [2].

In practice, we cannot achieve infinite resolution. To increase the likelihood of finding a globally optimal policy with a limited resolution of sampled states, we need an analog to exploration and global minimization with respect to actions found in the Bellman equation. We approximate this process by periodically reoptimizing each sampled state using the policies of other sampled states. As more and more states are sampled and many alternative sampled states are considered in optimizing any given sampled state, a wide range of actions are considered for each state. We run a reoptimization phase of the algorithm after every N (typically 100) states have been stored. There are several ways to design this reoptimization phase. Each state could use the policy of a nearest neighbor or a randomly chosen neighbor with the distribution being distance dependent or just choosing another state randomly with no consideration of distance (what we currently do). In [6], how to follow a policy of another sampled state if its trajectory is stored or can be recomputed as needed is described. In this paper, we explored a different approach that does not require each sampled state to save its trajectory or recompute it. To "follow" the policy of another state, we follow the locally linear policy for that state

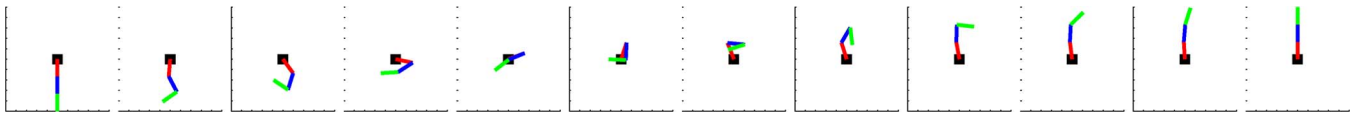

Fig. 5. Configurations from the simulated three-link pendulum optimal trajectory every tenth of a second and at the end of the trajectory.

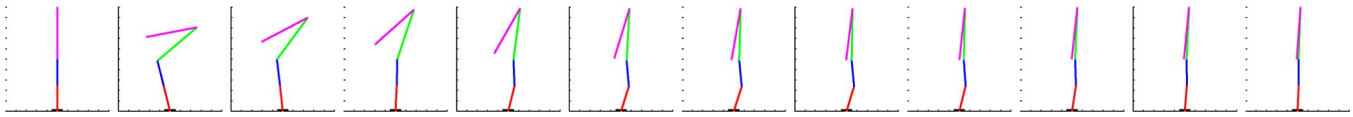

Fig. 6. Configurations every quarter second from a simulated response to a forward impulse (to the right) of 22.5 N · s. The lower black rectangle indicates the extent of the symmetric foot.

until the trajectory begins to go away from the state. At that point, we switch to following the globally approximated policy. Because we apply this reoptimization process periodically with different randomly selected policies, over time, we explore the use of a wide range of actions from each state. Our termination condition for the algorithm is to see if multiple runs of the algorithm get the same or similar answers.

IV. Results

In addition to the one-link swingup example presented in Section 1, we present results on two-link swingup (4-D state), three-link swingup (6-D state), and four-link balance (8-D state). For the one-link swingup case, the random state approach found a globally optimal trajectory (the same trajectory found by our grid-based approaches [50]) after adding only 63 random states. Fig. 3 shows the distribution of states and their trajectories superimposed on a contour map of the value function for one-link swingup, and Fig. 4 shows how the solved volume represented by the sampled states grows. For the two-link swingup case, the random state approach finds what we believe is a globally optimal trajectory (the same trajectory found by our tabular approaches [50]) after storing an average of 12 000 random states compared to 100 million states needed by a tabular approach. For the three-link swingup case, the random state approach found a good trajectory after storing less than 22 000 random states (Fig. 5). We were not able to solve this problem using regular grid-based approaches with a 4-GB table.

A. Simple Model of Standing Balance

We provide results on a standing robot balancer that is pushed (Fig. 6) to demonstrate that we can apply the approach to systems with 8-D states. This problem is hard because the ankle torque is quite limited to prevent the foot from tilting and the robot from falling. We created a four-link model that included a knee, shoulder, and arm. Each link is modeled as a thin rod, with a calf and thigh length of 0.5 meters and 17.5 kg each. The torso is 1 m long with a weight of 26.25 kg; the arm is 1 m long with a weight of 8.75 kg. We assume that, in standing, the center of pressure is at the center of the foot. We therefore use a symmetric foot that is 0.2 m long in our model. This results in a maximum ankle torque of approximately ± 70 N · m.

We model perturbations as horizontal impulses applied to the middle of the torso. The perturbations instantaneously change the joint velocities from zero to values appropriate for the perturbation: $\Delta\dot{\theta} = \mathbf{H}^{-1}\mathbf{J}^{\mathrm{T}} \int \text{force} \cdot dt$, where the elements of the inertia matrix $\mathbf{H}(\theta)$ are the coefficients of angular accelerations in the rigid body dynamics ($\boldsymbol{\tau} = \mathbf{H}\ddot{\boldsymbol{\theta}} + \ldots$), $\mathbf{J}(\theta)$ is the Jacobian matrix, and $\int \text{force} \cdot dt$ is the impulse applied (measured in Newton-seconds). We assume no slipping or other change of contact state during the perturbation.

The states are bounded by $-0.4 < \theta_a < 0.8$ rad, $-0.01 < \theta_k < 2.5$ rad, $-1.5 < \theta_h < 0.1$ rad, and $-0.5 < \theta_s < 2.5$ rad, where θ_a is the ankle angle, θ_k is the knee angle, θ_h is the hip angle, θ_s is the shoulder angle, and $\theta_i = 0$ is upright with the arms hanging down. The ankle torque is bounded by ± 70 N · m to keep the center of pressure within the foot. The knee and hip torques are bounded by ± 500 N · m. The shoulder torque is bounded by ± 250 N · m. The one-step optimization criterion is a combination of quadratic penalties on the deviations of the joint angles from their desired positions (straight up with the arm hanging down), the joint velocities, and the joint torques: $L(\mathbf{x}, \mathbf{u}) = (\theta_a^2 + \theta_k^2 + \theta_h^2 + \theta_s^2)T + (\dot{\theta}_a^2 + \dot{\theta}_k^2 + \dot{\theta}_h^2 + \dot{\theta}_s^2)T + 0.002(\tau_a^2 + \tau_k^2 + \tau_h^2 + \tau_s^2)T$, where 0.002 weights the torque penalty relative to the position and velocity errors and T is the time step of the simulation (0.01 s). The penalty on joint velocities reduces knee and shoulder oscillations. After dynamic programming based on approximately 60 000 sampled states, Fig. 6 shows the response to the largest perturbations that could be handled in the forward direction. We have designed an LQR controller that optimizes the same criterion on the four-link model. For perturbations of 17.5 N · s and higher, the LQR controller falls down, whereas the controller presented here is able to handle larger perturbations of 22.5 N · s.

V. Conclusion

We have combined random sampling of states, local models, and local trajectory optimization to create a promising approach to practical dynamic programming for robot control problems. We are able to solve problems that we could not solve before using tabular or global function approximation approaches. Future work will optimize aspects and variants of this approach and do a thorough comparison with alternative approaches.

References

[1] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[2] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 1995.
[3] C. G. Atkeson and S. Schaal, "Learning tasks from a single demonstration," in *Proc. IEEE ICRA*, 1997, pp. 1706–1712.
[4] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *Proc. 14th Int. Conf. Mach. Learn.*, 1997, pp. 12–20.
[5] C. G. Atkeson, "Nonparametric model-based reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 10. Cambridge, MA: MIT Press, 1998, pp. 1008–1014.
[6] C. G. Atkeson, "Using local trajectory optimizers to speed up global optimization in dynamic programming," in *Advances in Neural Information*

Processing Systems, vol. 6, J. D. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA: Morgan Kaufmann, 1994, pp. 663–670.
[7] F. L. Lewis and V. L. Syrmos, *Optimal Control*, 2nd ed. New York: Wiley-Interscience, 1995.
[8] I. Sloan and H. Wozniakowski, "When are quasi-Monte Carlo algorithms efficient for high dimensional integrals?" *J. Complex.*, vol. 14, no. 1, pp. 1–33, Mar. 1998.
[9] S. Li and W. K. Liu, *Meshfree Particle Methods*. New York: Springer-Verlag, 2004.
[10] J. Rust, "Using randomization to break the curse of dimensionality," *Econometrica*, vol. 65, no. 3, pp. 487–516, May 1997.
[11] C. Szepesvári, "Efficient approximate planning in continuous space Markovian decision problems," *AI Commun.*, vol. 13, no. 3, pp. 163–176, Sep. 2001.
[12] J. N. Tsitsiklis and V. B. Roy, "Regression methods for pricing complex American-style options," *IEEE Trans. Neural Netw.*, vol. 12, no. 4, pp. 694–703, Jul. 2001.
[13] V. D. Blondel and J. N. Tsitsiklis, "A survey of computational complexity results in systems and control," *Automatica*, vol. 36, no. 9, pp. 1249–1274, Sep. 2000.
[14] J. A. Boyan and A. W. Moore, "Generalization in reinforcement learning: Safely approximating the value function," in *Advances in Neural Information Processing Systems*, vol. 7, G. Tesauro, D. S. Touretzky, and T. K. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 369–376.
[15] R. Munos and C. Szepesvári, "Finite-time bounds for fitted value iteration," *JMLR*, vol. 9, pp. 815–857, 2008.
[16] O. Teytaud, S. Gelly, and J. Mary, "Active learning in regression, with application to stochastic dynamic programming," in *Proc. 4th ICINCO-ICSO*, 2007, pp. 198–205.
[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[18] M. Hauskrecht, "Incremental methods for computing bounds in partially observable Markov decision processes," in *Proc. 14th Nat. Conf. AAAI*, 1997, pp. 734–739.
[19] S. Thrun, "Monte Carlo POMDPs," in *Advances in Neural Information Processing*, vol. 12, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds. Cambridge, MA: MIT Press, 2000, pp. 1064–1070.
[20] N. L. Zhang and W. Zhang, "Speeding up the convergence of value iteration in partially observable Markov decision processes," *J. Artif. Intell. Res.*, vol. 14, pp. 29–51, 2001.
[21] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. IJCAI*, 2003, pp. 1025–1032.
[22] T. Smith and R. Simmons, "Heuristic search value iteration for POMDPs," in *Proc. Uncertainty Artif. Intell.*, 2004, pp. 520–527.
[23] M. T. J. Spaan and V. Nikos, "A point-based POMDP algorithm for robot planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, New Orleans, LA, Apr. 2004, pp. 2399–2404.
[24] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[25] R. Munos and A. W. Moore, "Variable resolution discretization in optimal control," *Mach. Learn.*, vol. 49, no. 2/3, pp. 291–323, Nov./Dec. 2002.
[26] R. L. Larson, *State Increment Dynamic Programming*. New York: Elsevier, 1968.
[27] J. J. Murray, C. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 32, no. 2, pp. 140–153, May 2002.
[28] R. L. Grossman, D. Valsamis, and X. Qin, "Persistent stores and hybrid systems," in *Proc. 32nd Conf. Decision Control*, 1993, pp. 2298–2302.
[29] C. G. Atkeson and J. Morimoto, "Non-parametric representation of a policies and value functions: A trajectory based approach," in *Advances in Neural Information Processing Systems*, vol. 15. Cambridge, MA: MIT Press, 2003.
[30] J. D. Schierman, D. G. Ward, J. R. Hull, N. Gandhi, M. W. Oppenheimer, and D. B. Doman, "Integrated adaptive guidance and control for re-entry vehicles with flight test results," *J. Guid. Control Dyn.*, vol. 27, no. 6, pp. 975–988, 2004.
[31] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Trans. Robot.*, vol. 21, no. 6, pp. 1077–1091, 2005.
[32] S. Ramamoorthy and B. J. Kuipers, "Qualitative hybrid control of dynamic bipedal walking," in *Proc. Robot.: Sci. Syst. Conf.*, 2006, pp. 89–96.
[33] M. Stolle and C. G. Atkeson, "Transfer of policies based on trajectory libraries," in *Proc. Int. Conf. IROS*, 2007, pp. 2981–2986.
[34] A. Safonova and J. K. Hodgins, "Construction and optimal search of interpolated motion graphs," in *Proc. SIGGRAPH*, Aug. 2007, vol. 26, no. 3.
[35] P. Werbos, 2007. personal communication.
[36] P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control*. New York: Academic, 1970.
[37] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. New York: Elsevier, 1970.
[38] E. Todorov, *Iterative Local Dynamic Programming*. [Online]. Available: http://www.cogsci.ucsd.edu/~todorov/papers.htm
[39] A. Altamimi, M. Abu-Khalaf, and F. L. Lewis, "Adaptive critic designs for discrete-time zero-sum games with application to H_∞ control," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 240–247, Feb. 2007.
[40] A. Altamimi, F. L. Lewis, and M. Abu-Khalaf, "Model-free q-learning designs for linear discrete-time zero-sum games with application to H-infinity control," *Automatica*, vol. 43, no. 3, pp. 473–481, Mar. 2007.
[41] J. Morimoto, G. Zeglin, and C. G. Atkeson, "Minimax differential dynamic programming: Application to a biped walking robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2003, pp. 1927–1932.
[42] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, II, *Handbook of Learning and Approximate Dynamic Programming*. New York: Wiley-IEEE Press, 2004.
[43] C. G. Atkeson and B. Stephens, "Random sampling of states in dynamic programming," in *Proc. NIPS Conf.*, 2007, to be published.
[44] C. G. Atkeson and B. Stephens, "Multiple balance strategies from one optimization criterion," in *Proc. IEEE-RAS Int. Conf. Humanoids*, 2007, to be published.
[45] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artif. Intell. Rev.*, vol. 11, no. 1–5, pp. 11–73, Feb. 1997.
[46] M. B. Abbott, *An Introduction to the Method of Characteristics*. London, U.K.: Thames & Hudson, 1966.
[47] R. Isaacs, *Differential Games*. New York: Dover, 1965.
[48] J. Lewin, *Differential Games*. New York: Springer-Verlag, 1994.
[49] M. Breitner, "Robust optimal on-board reentry guidance of a European space shuttle: Dynamic game approach and guidance synthesis with neural networks," in *Complex Dynamical Processes With Incomplete Information*, E. Reithmeier, Ed. Basel, Switzerland: Birkhauser, 1999.
[50] C. G. Atkeson, "Randomly sampling actions in dynamic programming," in *Proc. IEEE Int. Symp. ADPRL*, 2007, pp. 185–192.

Christopher G. Atkeson received the M.S. degree in applied mathematics (computer science) from Harvard University, Cambridge, MA, and the Ph.D. degree in brain and cognitive science from the Massachusetts Institute of Technology (MIT), Cambridge.

He joined the MIT faculty in 1986 and the Georgia Institute of Technology College of Computing, Atlanta, in 1994. He has been with Carnegie Mellon University (CMU), Pittsburgh, PA, since 2000, where he is currently a Professor with the Robotics Institute and the Human-Computer Interaction Institute. His research focuses on humanoid robotics and robot learning by using challenging dynamic tasks such as juggling. His specific research interests include nonparametric learning, memory-based learning including approaches based on trajectory libraries, reinforcement learning, and other forms of learning based on optimal control, learning from demonstration, and modeling human behavior.

Dr. Atkeson has received a National Science Foundation Presidential Young Investigator Award, a Sloan Research Fellowship, and a Teaching Award from the MIT Graduate Student Council.

Benjamin J. Stephens received the B.Sc. degree in mechanical engineering from Northwestern University, Evanston, IL, in 2006. He is currently working toward the Ph.D. degree in robotics at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

His research at the Robotics Institute focuses on the control of humanoid robots during dynamic balance and locomotion tasks.