Trajectory-Based Dynamic Programming

Christopher G. Atkeson and Chenggang Liu

Abstract. We informally review our approach to using trajectory optimization to accelerate dynamic programming. Dynamic programming provides a way to design globally optimal control laws for nonlinear systems. However, the curse of dimensionality, the exponential dependence of memory and computation resources needed on the dimensionality of the state and control, limits the application of dynamic programming in practice. We explore trajectory-based dynamic programming, which combines many local optimizations to accelerate the global optimization of dynamic programming. We are able to solve problems with less resources than grid-based approaches, and to solve problems we couldn't solve before using tabular or global function approximation approaches.

1 What Is Dynamic Programming?

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 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 $L(\mathbf{x}, \mathbf{u})$, so we use discrete time dynamic programming. We are focusing on steady state policies and thus an infinite time horizon. Action vectors are typically limited to a finite volume set.

Christopher G. Atkeson
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
e-mail: cga@cmu.edu

Chenggang Liu
Department of Automation, Shanghai Jiao Tong University, Shanghai, China
e-mail: cgliu2008@gmail.com

K. Mombaur and K. Berns (Eds.): *Modeling, Simulation and Optimization*, COSMOS 18, pp. 1–15.
DOI: 10.1007/978-3-642-36368-9_1 © Springer-Verlag Berlin Heidelberg 2013

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 $\mathbf{x}_j$ until the value function estimates have converged. Typically the value function and control law are represented on a regular grid. Some type of interpolation is used to approximate these functions within each grid cell. If each dimension of the state and action is represented with a resolution R, and the dimensionality of the state is d_x and that of the action is d_u, the computational cost of the conventional approach is proportional to $R^{d_x} \times R^{d_u}$ and the memory cost is proportional to R^{d_x}. This exponential dependence of cost on dimensionality is known as the Curse of Dimensionality [1].

An example problem: We use one link pendulum swingup as an example problem to provide the reader with a visualizable example of a nonlinear control law and corresponding value function. 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 a one step cost function penalizes the amount of torque used and the deviation of the current angle 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 angle errors (θ: difference between current angle and the goal angle) and the squared torques τ: $L(\mathbf{x},\mathbf{u}) = 0.1\theta^2 + \tau^2$ where 0.1 weights the angle error relative to the torque penalty. There are no costs associated with the joint velocity. The uniform density link has a mass m of 1kg, length l of 1m, and width of 0.1m. The dynamics are given by:

$$\ddot{\theta} = \frac{(\tau + 0.5m \cdot g \cdot l \cdot \sin(\theta))}{\mathbf{I}} \tag{1}$$

where g is the gravitational constant 9.81 and $\mathbf{I}$ is the moment of inertia about the hinge. The continuous time dynamics are discretized with a time step of 0.01s using Euler's method as discrete time dynamics are more convenient for system identification and computer-based discrete time control. Because the dynamics and cost function are time invariant, there is a steady state control law and value function (Fig. 2). Because we keep track of the direction of the error and multiple rotations around the hinge, there is a unique optimal trajectory. In general there may be multiple solutions with equal optimal costs. Dynamic programming converges to one of the globally optimal solutions.

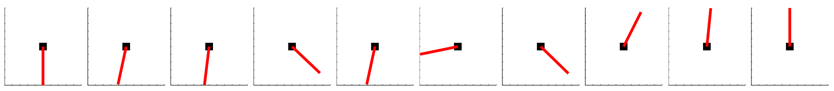

Fig. 1 Configurations from the simulated one link pendulum swingup optimal trajectory every half second and at the end of the trajectory. The pendulum starts in the downward position (left) and swings up in rightward configurations.

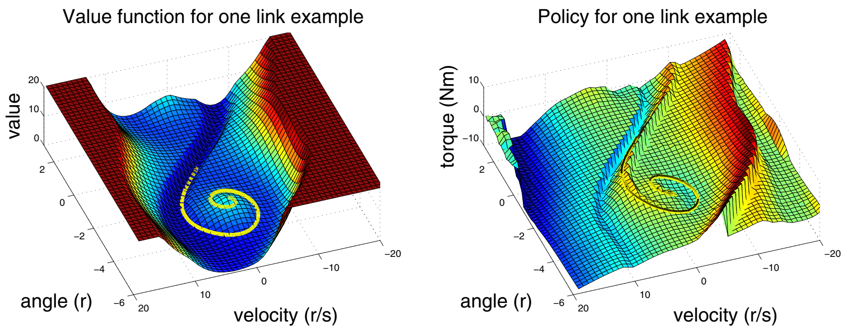

Fig. 2 The value function and policy for a one link pendulum swingup. The optimal trajectory is shown as a line in the value function and policy plots. The value function is cut off above 20 so we can see the details of the part of the value function that determines the optimal trajectory. The goal is the state (0,0), upright and not moving.

Representing trajectories explicitly to achieve representational sparseness: A technique to accelerate dynamic programming is to optimize more than one step at a time. Larson proposed modifying the Bellman equation to allow multiple time steps and multiple evaluations of the one step cost and dynamics before evaluating the value function on the right hand side [3]:

$$V(\mathbf{x}_0) = \min_{\mathbf{u}_{0,N-1}} \left(\left(\sum_{0}^{N-1} L(\mathbf{x}_i, \mathbf{u}_i) \right) + V(\mathbf{x}_N) \right) \tag{2}$$

In a grid-based approximation with multilinear interpolation, $V(\mathbf{x})$ depends on the value estimates at all the surrounding nodes. Larson's goal was to ensure that $V(\mathbf{x}_N)$ on the right hand side of the Bellman equation did not depend on the value being updated $(V(\mathbf{x}_0))$ by ensuring that the trajectory ended far enough away from its start in his State Increment Dynamic Programming. We have extended this idea by running trajectories a variety of distances including all the way to the goal. To help show that representing trajectories explicitly allows greater sparseness in dynamic programming, we show its effect on the one link swingup task. Fig. 3-top-left shows Larson's State Increment Dynamic Programming procedure on a 10x10 grid applied to this problem. In Larson's approach trajectories are run until they exit a 2x2 volume and the start value has no effect on the end value when multi-linear interpolation is used on the grid of values. Fig. 3-top-right shows a set of optimized trajectories that run all the way to the goal from a similar grid. The flow from state to state is clearly indicated. When the resolution is greatly reduced, the State Increment Dynamic Programming approach fails (Fig. 3-bottom-left), while the full trajectory-based approach is more robust to the sparse representation (Fig. 3-bottom-right) and still generates globally optimal trajectories. This work raises the question: "What should the length of the trajectory be?" Larson used a distance threshold. We used reaching the goal (attaining a point with zero future costs) as a threshold. A time

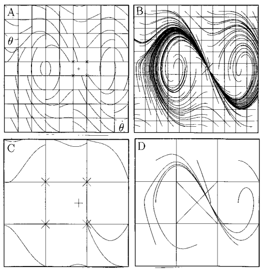

Fig. 3 Right: Different approaches to computing and representing the value function for one link swingup. On the left is the State Increment Dynamic Programming Approach of Larson. On the right trajectories are run all the way to the goal. The plots are of phase space with angles on the x axis and angular velocities on the y axis.

threshold could also be used. What distance or time threshold value should be used? Should it be the same throughout the space? Another question is how to efficiently optimize the sequence of actions in Eq. 2. We use local trajectory optimization to find an optimal sequence of actions.

2 Trajectory-Based Dynamic Programming

Our approach modifies (and complements) existing approximate dynamic programming approaches in a number of ways: 1) We approximate the value function and policy using many local models (quadratic for the value function, linear for the policy) as shown in Fig. 4. These local models, located at sampled states, help our function approximators handle sparsely sampled states. A nearest neighbor approach is taken to determine which local model should be used to predict the value and policy for a particular state. 2) We use trajectory segments rather than single time steps to perform Bellman updates (black lines in Fig. 4-Right). 3) After using either the approximated policy or value function to initialize the trajectory segment, we use trajectory optimization to directly optimize the sequence of actions $\mathbf{u}_{0,N-1}$ and the corresponding states $\mathbf{x}_{1,N}$. 4) Local models of the value function and policy are created as a byproduct of our trajectory optimization process. 5) Local models exchange information to ensure the Bellman equation is satisfied everywhere and the value function and policy are globally optimal. 6) We also use trajectory optimization on each query to refine the predicted values and actions. 7) We are exploring using adaptive grids. Fig. 4-Right shows a randomly generated set of states superimposed on a contour plot of the value function for one link swingup, and the optimized trajectories used to generate locally quadratic value function models.

Local models of the value function and policy: We need to represent value functions and policies sparsely. We use 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

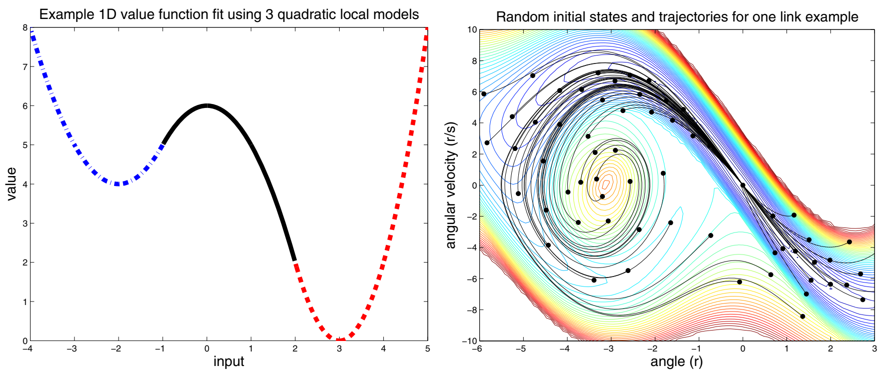

Fig. 4 **Left:** Example of a local approximation of a 1D value function using three quadratic models. **Right:** Random states (dots) used to plan one link swingup, superimposed on a contour map of the value function. Optimized trajectories (black lines) are shown starting from the random states.

of a function at different points. At each sampled state $\mathbf{x}^p$ the local quadratic model for the value function is:

$$V^p(\mathbf{x}) = V_0^p + \mathbf{V}_x^p\hat{\mathbf{x}} + \frac{1}{2}\hat{\mathbf{x}}^{\mathrm{T}}\mathbf{V}_{xx}^p\hat{\mathbf{x}} \tag{3}$$

where $\hat{\mathbf{x}} = \mathbf{x} - \mathbf{x}^p$ is the vector from the sampled state $\mathbf{x}^p$ to the query $\mathbf{x}$, V_0^p is the constant term, $\mathbf{V}_x^p$ is the first derivative with respect to state at $\mathbf{x}^p$, and $\mathbf{V}_{xx}^p$ is the second spatial derivative at $\mathbf{x}^p$. The local linear model for the policy is:

$$\mathbf{u}^p(\mathbf{x}) = \mathbf{u}_0^p - \mathbf{K}^p\hat{\mathbf{x}} \tag{4}$$

where $\mathbf{u}_0^p$ is the constant term, and $\mathbf{K}^p$ is the first derivative of the local policy with respect to state at $\mathbf{x}^p$ 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.

Creating the local models: These local models are created using Differential Dynamic Programming (DDP) [4, 5, 6, 7]. This local trajectory optimization process is similar to linear quadratic regulator design in that a value function and policy is produced. In DDP, value function and policy models are produced at each point along a trajectory. Suppose at a time step i we have 1) a local second order Taylor series approximation of the optimal value function: $V^i(\mathbf{x}) = V_0^i + \mathbf{V}_x^i\hat{\mathbf{x}} + \frac{1}{2}\hat{\mathbf{x}}^{\mathrm{T}}\mathbf{V}_{xx}^i\hat{\mathbf{x}}$ where $\hat{\mathbf{x}} = \mathbf{x} - \mathbf{x}^i$. 2) a local second order Taylor series approximation of the robot dynamics ($\mathbf{f}_x^i$ and $\mathbf{f}_u^i$ correspond to $\mathbf{A}$ and $\mathbf{B}$ of the linear plant model used in linear quadratic regulator (LQR) design): $\mathbf{f}^i(\mathbf{x},\mathbf{u}) = \mathbf{f}_0^i + \mathbf{f}_x^i\hat{\mathbf{x}} + \mathbf{f}_u^i\hat{\mathbf{u}} + \frac{1}{2}\hat{\mathbf{x}}^{\mathrm{T}}\mathbf{f}_{xx}^i\hat{\mathbf{x}} + \hat{\mathbf{x}}^{\mathrm{T}}\mathbf{f}_{xu}^i\hat{\mathbf{u}} + \frac{1}{2}\hat{\mathbf{u}}^{\mathrm{T}}\mathbf{f}_{uu}^i\hat{\mathbf{u}}$ 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 $\mathbf{R}$ of LQR design): $L^i(\mathbf{x},\mathbf{u}) = L_0^i + \mathbf{L}_x^i\hat{\mathbf{x}} + \mathbf{L}_u^i\hat{\mathbf{u}} + \frac{1}{2}\hat{\mathbf{x}}^{\mathrm{T}}\mathbf{L}_{xx}^i\hat{\mathbf{x}} + \hat{\mathbf{x}}^{\mathrm{T}}\mathbf{L}_{xu}^i\hat{\mathbf{u}} + \frac{1}{2}\hat{\mathbf{u}}^{\mathrm{T}}\mathbf{L}_{uu}^i\hat{\mathbf{u}}$

Given a trajectory, one can integrate the value function and its first and second spatial derivatives backwards in time to compute an improved value function and policy. We utilize the "Q function" notation [35] 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):

$$\mathbf{Q}^i_x = \mathbf{L}^i_x + \mathbf{V}^i_x \mathbf{f}^i_x; \quad \mathbf{Q}^i_u = \mathbf{L}^i_u + \mathbf{V}^i_x \mathbf{f}^i_u \tag{5}$$

$$\mathbf{Q}^i_{xx} = \mathbf{L}^i_{xx} + \mathbf{V}^i_x \mathbf{f}^i_{xx} + (\mathbf{f}^i_x)^{\mathrm{T}} \mathbf{V}^i_{xx} \mathbf{f}^i_x \tag{6}$$

$$\mathbf{Q}^i_{ux} = \mathbf{L}^i_{ux} + \mathbf{V}^i_x \mathbf{f}^i_{ux} + (\mathbf{f}^i_u)^{\mathrm{T}} \mathbf{V}^i_{xx} \mathbf{f}^i_x \tag{7}$$

$$\mathbf{Q}^i_{uu} = \mathbf{L}^i_{uu} + \mathbf{V}^i_x \mathbf{f}^i_{uu} + (\mathbf{f}^i_u)^{\mathrm{T}} \mathbf{V}^i_{xx} \mathbf{f}^i_u \tag{8}$$

$$\Delta \mathbf{u}^i = (\mathbf{Q}^i_{uu})^{-1} \mathbf{Q}^i_u; \quad \mathbf{K}^i = (\mathbf{Q}^i_{uu})^{-1} \mathbf{Q}^i_{ux} \tag{9}$$

$$\mathbf{V}^{i-1}_x = \mathbf{Q}^i_x - \mathbf{Q}^i_u \mathbf{K}^i; \quad \mathbf{V}^{i-1}_{xx} = \mathbf{Q}^i_{xx} - \mathbf{Q}^i_{xu} \mathbf{K}^i \tag{10}$$

where subscripts indicate derivatives and superscripts indicate the trajectory index. After the backward sweep, forward integration can be used to update the trajectory itself: $\mathbf{u}^i_{new} = \mathbf{u}^i - \Delta \mathbf{u}^i - \mathbf{K}^i(\mathbf{x}^i_{new} - \mathbf{x}^i)$. We note that the cost of this approach grows at most cubically rather than exponentially with respect to the dimensionality of the state. We formulate the trajectory optimization with an infinite time horizon so that the value functions and control laws are time invariant and functions only of state.

Combining greedy local optimizers to perform global optimization: As currently described, the algorithm finds a locally optimal policy, but not necessarily a globally optimal policy. However, if the combination of local value function models generate a global value function that satisfies the Bellman equation everywhere, the resulting policy and value function are globally optimal [1, 2]. We will refer to violations of the Bellman equation as "Bellman errors". We can reduce one step Bellman errors

$$e = V(\mathbf{x}) - \min_{\mathbf{u}}(L(\mathbf{x},\mathbf{u}) + V(\mathbf{f}(\mathbf{x},\mathbf{u}))) \tag{11}$$

and multi-step Bellman errors

$$e = V(\mathbf{x}_0) - \min_{\mathbf{u}_{0,N-1}} \left(\left(\sum_{0}^{N-1} L(\mathbf{x}_i,\mathbf{u}_i) \right) + V(\mathbf{x}_N) \right) \tag{12}$$

by 1) re-optimizing local models that disagree using policies from neighboring local models, and 2) adding additional local models in the area of the discrepancies until Bellman errors are reduced below a threshold everywhere (up to a sampling resolution). This process does require globally optimizing the one step action $\mathbf{u}$ or multi-step action sequence $\mathbf{u}_{0,N-1}$ for each test. The Bellman error 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]. A weaker test which verifies that the value function matches the current policy assesses the Bellman error for $\mathbf{u}(\mathbf{x})$ at each selected state, so no global minimization is necessary. This test is useful in policy iteration.

A useful heuristic to detect local optima that does not require a global optimization on each test is to enforce continuity of the value function and the policy. This heuristic often works because a switch from a global optimum to a local optimum in a policy often shows up as a discontinuity in the policy or value function. Unfortunately, often optimal policies and value functions have true discontinuities. As Fig. 2 shows, value functions can have derivative discontinuities (discontinuities of the spatial derivatives of the value, see the creases in the figure) at policy discontinuities. In addition, value functions can have discontinuities of the value itself in complex situations such as when there are multiple goals (zero velocity states that require no cost to maintain) and it is not possible to reach all goals from each state. A second heuristic is that optimal trajectories should not normally cross any policy or value function discontinuities given smooth dynamics and one step cost functions. However, there are exceptions to this heuristic as well.

Discrepancies between predictions of local value functions can also be used to guide computational effort and allocate local models. Discrepancies of local policies can be considered by using the local policies to generate trajectory segments, and seeing if the cost of the trajectory is accurately predicted by local value function models. We can enforce continuity of local models by 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 local models in between nearest neighbors that continue to disagree until the discontinuity is confirmed or eliminated [6]. We also periodically reoptimize each local model using the policies of other local models. As more neighboring policies are considered in optimizing any given local model, a wide range of actions are considered for each state. There are several ways to perform reoptimization. Each local model could use the policy of a nearest neighbor, or a randomly chosen neighbor with the distribution being distance dependent, or just choosing another local model randomly with no consideration of distance. [6] describes how to follow a policy of another sampled state if its trajectory is stored, or can be recomputed as needed. We have also 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 until the trajectory begins to go away from the state. At that point we switch to following the globally approximated policy. Since we apply this reoptimization process periodically with different randomly selected local models, over time we explore using a wide range of actions from each state. This process is analogously to exploration in learning and to the global minimization with respect to actions found in the Bellman equation. This approach is similar to using the method of characteristics to solve partial differential equations [8] and finding value functions for games [9, 10, 11]. We note that value functions that are discontinuous in known locations, with known patterns, or in a relatively small area can also be handled with approaches that partition the space into regions with no discontinuities.

Adaptive grids — constant value contours: We have explored a number of adaptive grid techniques for trajectory-based dynamic programming. Adaptive grid techniques for solving partial differential equations are useful for dynamic programming as well [12]. Fig. 5 shows a trajectory-based approach being used to compute a

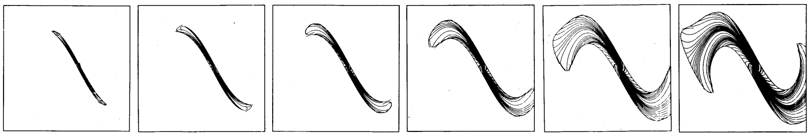

Fig. 5 Computing a 1D swingup value function using an adaptive grid. The plots are of phase space with angles on the x axis and angular velocities on the y axis.

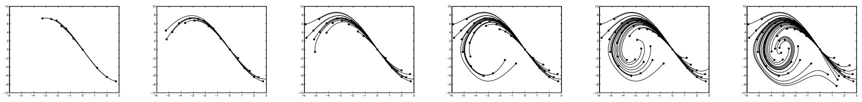

Fig. 6 Randomly sampled states and trajectories for the one link swingup problem after 10, 20, 30, 40, 50, and 60 states are stored. These figures correspond to Figs. 4:right and 5, with angle on the x axis and angular velocity on the y axis.

global value function [6, 7]. An adaptive grid of initial conditions are maintained on a "frontier" of constant value $V(\mathbf{x})$ or cost-to-go. This "frontier" is one dimension less than the dimensionality of $\mathbf{x}$. Trajectories are optimized from each sample of the frontier and local models are maintained at each sample. The value function at each frontier sample is compared with that of nearby points, using the local models for the value functions and policies. At discrepancies the trajectories are re-optimized using the value function from the neighboring frontier point. If this fails to resolve the discrepancy, new frontier points are added at the discrepancy until the discrepancy is below a threshold. Fig. 5 shows the frontier being gradually expanded. Since each trajectory optimization is independent, these approaches are "embarrassingly" parallel.

Adaptive grids — randomly sampling states: Fig. 6 shows an adaptive grid approach based on randomly sampling states, similar to Fig. 5. In this case states are randomly sampled. If the predicted value V (using the nearest local model) for a state is too high, it is rejected. If the predicted value is too similar to the cost of an optimized trajectory, it is rejected. Otherwise it is added to the database of sampled states, with its local value function and policy models. To generate the initial trajectory for optimization the current approximated policy is used 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 trajectory is then locally optimized.

We solve a series of problems by gradually increasing the cost of trajectories we consider. Each cost threshold generates a volume we consider, and in the most conservative version of our algorithms, we completely solve each volume before increasing the cost threshold. More aggresive versions only partially solve each volume before increasing the cost threshold, and continue to update lower cost nodes throughout execution.

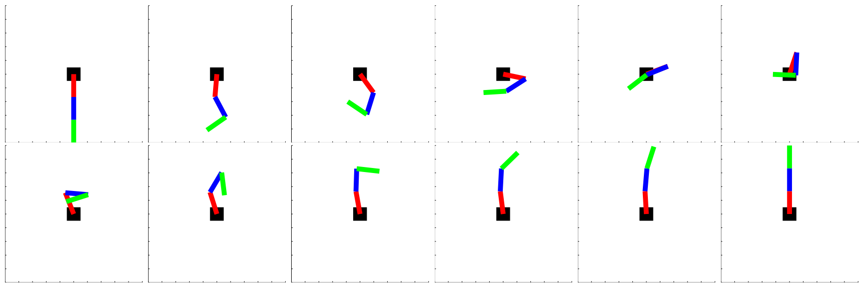

Fig. 7 Configurations from the simulated three link pendulum optimal swingup trajectory every tenth of a second and at the end of the trajectory

We expect the locally optimal policies to be fairly good because we 1) gradually increase the solved volume (Fig. 6) and 2) use local optimizers. Given local optimization of actions, gradually increasing the solved volume defined by a constant value contour will result in a globally optimal policy if the boundary of this volume never touches a non-adjacent section of itself, given reasonable dynamics and one step cost functions. Fig. 2 and 4 show the creases in the value function (discontinuities in the spatial derivative) and corresponding discontinuities in the policy that typically result when the constant value contour touches a non-adjacent section of itself as the limit on acceptable values is increased.

3 Results

In addition to the one link swingup example presented in the introduction, we present results on two link swingup (4 dimensional state), three link swingup (6 dimensional state), four link balance (8 dimensional state), and 5 link bipedal walking (10 dimensional state). In the first four cases we used a random adaptive grid approach [13]. For the one link swingup case, the random state approach found a globally optimal trajectory (the same trajectory found by our grid based approaches [14]) after adding only 63 random states. Fig. 4 shows the distribution of states and their trajectories superimposed on a contour map of the value function for one link swingup and Fig. 6 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 [14]) after storing an average of 12000 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 22000 random states (Fig. 7). We were not able to solve this problem using regular grid-based approaches with a 4 gigabyte table.

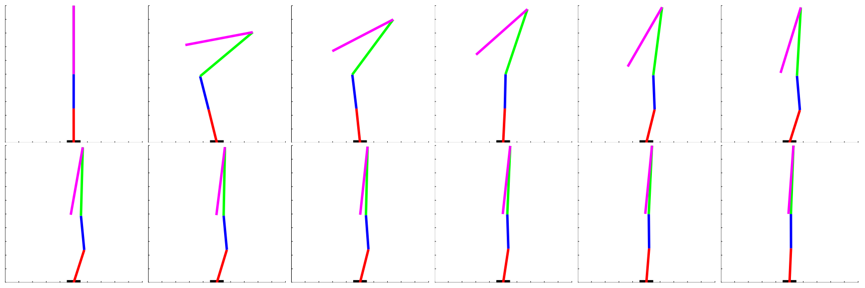

Fig. 8 Configurations every quarter second from a simulated response to a forward push (to the right) of 22.5 Newton-seconds. The lower black rectangle indicates the extent of the symmetric foot.

A simple model of standing balance: We provide results on a standing robot balancer that is pushed (Fig. 8), to demonstrate that we can apply the approach to systems with eight dimensional states. This problem is hard because the ankle torque is quite limited to prevent the foot from tilting and the robot falling. We created a four link model that included a knee, shoulder, and arm. Each link is modeled as a thin rod. 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. We assume no slipping or other change of contact state during the perturbation. Both the allowable states and possible torques are limited. 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: $L(\mathbf{x},\mathbf{u}) = (\theta_a^2 + \theta_k^2 + \theta_h^2 + \theta_s^2) + (\dot{\theta}_a^2 + \dot{\theta}_k^2 + \dot{\theta}_h^2 + \dot{\theta}_s^2) + 0.002(\tau_a^2 + \tau_k^2 + \tau_h^2 + \tau_s^2)$ where 0.002 weights the torque penalty relative to the position and velocity errors. The penalty on joint velocities reduces knee and shoulder oscillations. After dynamic programming based on approximately 60,000 sampled states, Fig. 8 shows the response to the largest perturbations that could be handled in the forward direction. We have designed a linear quadratic regulator (LQR) controller that optimizes the same criterion on the four link model, using a linearized dynamic model. For perturbations of 17.5 Newton-seconds and higher, the LQR controller falls down, while the controller presented here is able to handle larger perturbations of 22.5 Newton-seconds. We were able to generate behavior using optimization that matched human responses for large perturbations [15, 16]. Interestingly, we found that a single optimization criterion generated multiple strategies (both an ankle and hip strategy, for example).

We explored trajectory-based control of bipedal walking. We simulated a 5 link planar robot (2 legs and a torso). We optimized a periodic steady state trajectory (solid line) and 12 additional optimal trajectory segments starting just after -4 and 10 Newton-seconds perturbations at the hip at different times (Figure 9-left). The trajectory library was evaluated using perturbations of -10, -6, 6, 16, and 20 Newton-seconds at the hip (Figure 9-right). The robot successfully recovered from these

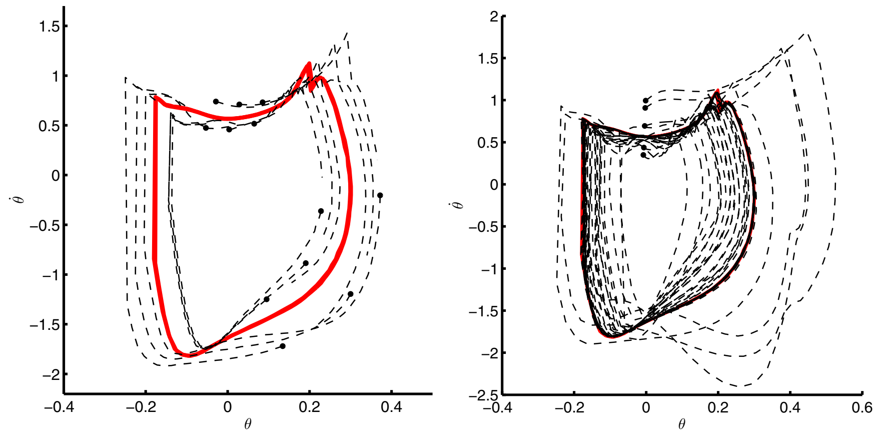

Fig. 9 Trajectory-based dynamic programming applied to bipedal walking. On the left we show the entries in a trajectory library, and on the right we show trajectories generated from the trajectory library in response to perturbations. The solid curve is the periodic steady state trajectory. 2D phase portraits are shown which are projections of the actual 10D trajectories. We plot the angle (x axis) and angular velocity (y axis) of a line from the hip to a foot.

perturbations. The simulated robot could also walk up and down 5 degree inclines using this trajectory-based policy generated by optimizing walking on level ground.

4 Related Work

Trajectories: In our approach we use trajectories to provide a more accurate estimate of the value of a state. In reinforcement learning "rollout" or simulated trajectories are often used to provide training data for approximating value functions [17, 18], as well as evaluating expectations in stochastic dynamic programming. Murray et. al. used trajectories to provide estimates of values of a set of initial states [19]. A number of efforts have been made to use collections of trajectories to represent policies [3, 6, 7, 20, 21, 22, 23, 24, 25, 26, 27]. [21] created sets of locally optimized trajectories to handle changes to the system dynamics. NTG uses trajectory optimization based on trajectory libraries for nonlinear control [28]. [6] and [7] used information transfer between stored trajectories to form sets of globally optimized trajectories for control.

Local models: We use local models of the value function and policy. Werbos proposed using local quadratic models of the value function [29]. 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 [4, 5]. Similar trajectory optimization approaches could have been used [30], including robust trajectory optimization approaches [31, 32, 33]. An alternative to local value function and policy models are global parametric models, for example [17, 34, 35]. 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.

5 Discussion

On what problems will our approach work well? We believe 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 [36]. 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 local 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 our work 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 our work 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.

Why dynamic programming? To generate a control law or policy, trajectory optimization can be applied to many initial conditions, and the resulting actions can be interpolated as needed. If trajectory optimization is fast enough it can be done online, as in Receding Horizon Control/Model Predictive Control (RHC/MPC). Why do we need to deal with dynamic programming and the curse of dimensionality? Dynamic programming is a global optimizer, while trajectory optimization alone finds local optima. Often, the local optima found using just trajectory optimization are not acceptable.

What about state estimation, learning models, and robust policies? We assume 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 [37, 38, 39].

Aren't there better trajectory optimization methods than DDP? DDP, invented in the 1960s, is useful because it produces local models of value functions and policies. It may be the case that newer methods can optimize trajectories faster than

DDP, and that we can use a combination of methods to achieve our goals. Parametric trajectory optimization based on sequential quadratic programming (SQP) dominates work in aerospace and animation. We have used SQP methods to initially optimize trajectories, and a final pass of DDP to produce local models of value functions and policies.

6 Future Work

Future work will optimize aspects and variants of this approach and do a thorough comparison with alternative approaches. More extensive experimentation will lead to a clearer understanding of when this approach works well, and how much storage and computation costs are reduced in general. An interesting but difficult research question is how sacrificing global optimality would enable finding useful solutions to bigger problems. Another interesting question is how to combine Receding Horizon Control/Model Predictive Control with a pre-computed value function [40, 41].

From our point of view, the most important question is whether model-based optimal control of this form can be usefully applied to humanoid robots, where the dynamics and thus the model depend on a poorly characterized environment as well as a well characterized robot.

7 Conclusion

We have combined local models and local trajectory optimization to create a promising approach to practical dynamic programming for robot control problems. New elements in our work relative to other trajectory library approaches include variable-length trajectories including trajectories all the way to a goal, using local models of the value function and policy, and maintaining consistency across local models of the value function. We are able to solve problems with less resources than grid-based approaches, and to solve problems we couldn't solve before using tabular or global function approximation approaches.

Acknowledgements. This material is based upon work supported by a National Natural Science Foundation of China Key Project (Grant No. 60935001) and in part by the US National Science Foundation (Grants EEC-0540865, ECCS-0824077, and IIS-0964581) and the DARPA M3 program.

References

1. Bellman, R.: Dynamic Programming (1957); reprinted by Dover 2003
2. Bertsekas, D.P.: Dynamic Programming and Optimal Control. Athena Scientific (1995)
3. Larson, R.L.: State Increment Dynamic Programming. Elsevier, New York (1968)

4. Dyer, P., McReynolds, S.R.: The Computation and Theory of Optimal Control. Academic Press, New York (1970)
5. Jacobson, D.H., Mayne, D.Q.: Differential Dynamic Programming. Elsevier, New York (1970)
6. Atkeson, C.G.: Using local trajectory optimizers to speed up global optimization in dynamic programming. In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) Advances in Neural Information Processing Systems, vol. 6, pp. 663–670. Morgan Kaufmann Publishers, Inc. (1994)
7. Atkeson, C.G., Morimoto, J.: Non-parametric representation of a policies and value functions: A trajectory-based approach. In: Advances in Neural Information Processing Systems, vol. 15. MIT Press (2003)
8. Abbott, M.B.: An Introduction to the Method of Characteristics. Thames & Hudson (1966)
9. Isaacs, R.: Differential Games. Dover (1965)
10. Lewin, J.: Differential Games. Spinger (1994)
11. Breitner, M.: Robust optimal on-board reentry guidance of a European space shuttle: Dynamic game approach and guidance synthesis with neural networks. In: Reithmeier, E. (ed.) Complex Dynamical Processes with Incomplete Information. Birkhauser, Basel (1999)
12. Munos, R.: Munos home,
`http://www.researchers.lille.inria.fr/~munos/` (2006)
13. Atkeson, C.G., Stephens, B.: Random sampling of states in dynamic programming. IEEE Transactions on Systems, Man, and Cybernetics, Part B 38(4), 924–929 (2008)
14. Atkeson, C.G.: Randomly sampling actions in dynamic programming. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL (2007)
15. Atkeson, C.G., Stephens, B.: Multiple balance strategies from one optimization criterion. In: IEEE-RAS International Conference on Humanoid Robots, Humanoids (2007)
16. Stephens, B.: Integral control of humanoid balance. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2007)
17. Boyan, J.A., Moore, A.W.: Generalization in reinforcement learning: Safely approximating the value function. In: Tesauro, G., Touretzky, D.S., Leen, T.K. (eds.) Advances in Neural Information Processing Systems, vol. 7, pp. 369–376. The MIT Press, Cambridge (1995)
18. Tsitsiklis, J.N., Van Roy, B.: Regression methods for pricing complex American-style options. IEEE-NN 12, 694–703 (2001)
19. Murray, J.J., Cox, C., Lendaris, G.G., Saeks, R.: Adaptive dynamic programming. IEEE Transactions on Systems, Man. and Cybernetics, Part C: Applications and Reviews 32(2), 140–153 (2002)
20. Grossman, R.L., Valsamis, D., Qin, X.: Persistent stores and hybrid systems. In: Proceedings of the 32nd Conference on Decision and Control, pp. 2298–2302 (1993)
21. Schierman, J.D., Ward, D.G., Hull, J.R., Gandhi, N., Oppenheimer, M.W., Doman, D.B.: Integrated adaptive guidance and control for re-entry vehicles with flight test results. Journal of Guidance, Control, and Dynamics 27(6), 975–988 (2004)
22. Frazzoli, E., Dahleh, M.A., Feron, E.: Maneuver-based motion planning for nonlinear systems with symmetries. IEEE Transactions on Robotics 21(6), 1077–1091 (2005)
23. Ramamoorthy, S., Kuipers, B.J.: Qualitative hybrid control of dynamic bipedal walking. In: Proceedings of the Robotics: Science and Systems Conference, pp. 89–96. MIT Press, Cambridge (2006)

24. Stolle, M., Tappeiner, H., Chestnutt, J., Atkeson, C.G.: Transfer of policies based on trajectory libraries. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2007)
25. Safonova, A., Hodgins, J.K.: Construction and optimal search of interpolated motion graphs. In: SIGGRAPH (2007)
26. Tedrake, R.: LQR-Trees: Feedback motion planning on sparse randomized trees. In: Proceedings of Robotics: Science and Systems (RSS), p. 8 (2009)
27. Reist, P., Tedrake, R.: Simulation-based LQR-trees with input and state constraints. In: IEEE International Conference on Robotics and Automation, ICRA (2010)
28. Milam, M., Mushambi, K., Murray, R.: NTG - a library for real-time trajectory generation (2002), `http://www.cds.caltech.edu/murray/software/2002antg.html`
29. Werbos, P.: Personal communication (2007)
30. Todorov, E., Tassa, Y.: Iterative local dynamic programming. In: 2nd IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), pp. 90–95 (2009)
31. Altamimi, A., Abu-Khalaf, M., Lewis, F.L.: Adaptive critic designs for discrete-time zero-sum games with application to H-infinity control. IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics 37(1), 240–247 (2007)
32. Altamimi, A., Lewis, F.L., Abu-Khalaf, M.: Model-free Q-learning designs for linear discrete-time zero-sum games with application to H-infinity control. Automatica 43, 473–481 (2007)
33. Morimoto, J., Zeglin, G., Atkeson, C.G.: Minmax differential dynamic programming: Application to a biped walking robot. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2003)
34. Si, J., Barto, A.G., Powell, W.B., Wunsch II, D.: Handbook of Learning and Approximate Dynamic Programming. Wiley-IEEE Press (2004)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
36. Lewis, F.L., Syrmos, V.L.: Optimal Control, 2nd edn. Wiley Interscience (1995)
37. Atkeson, C.G., Schaal, S.: Learning tasks from a single demonstration. In: Proceedings of the 1997 IEEE International Conference on Robotics and Automation (ICRA 1997), pp. 1706–1712 (1997)
38. Atkeson, C.G., Schaal, S.: Robot learning from demonstration. In: Proc. 14th International Conference on Machine Learning, pp. 12–20. Morgan Kaufmann (1997)
39. Atkeson, C.G.: Nonparametric model-based reinforcement learning. In: Advances in Neural Information Processing Systems, vol. 10, pp. 1008–1014. MIT Press, Cambridge (1998)
40. Liu, C., Su, J.: Biped walking control using offline and online optimization. In: 30th Chinese Control Conference (2011)
41. Tassa, Y., Erez, T., Todorov, E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2012)