Locally Optimal Planning in High Dimensions

Kei Usui

CMU-RI-TR-07-34

August 2007

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Master of Science

Abstract

Algorithms which guarantee to find globally optimal trajectories by covering a hypervolume of state space become infeasible in high dimensions. We have developed an algorithm which generates locally optimal trajectories without covering a hypervolume which only requires polynomial memory and time. Even though the algorithm does not guarantee a globally optimal trajectory in a deterministic manner, it is experimentally shown that resulting trajectories are reasonable for pendulum swing-up tasks. Results are shown for pendulum with 1- and 2-links and compared to globally optimal trajectories obtained by dynamic programming. Furthermore, results for 3-link pendulum of which globally optimal trajectory is currently unknown are presented.

Advisor: Christopher G. Atkeson

CONTENTS

I. INTRODUCTION

Globally optimal trajectories can be obtained using algorithms such as dynamic programming (DP) or A* for discrete domains. A standard approach to continuous domains using these approaches is to discretize sufficient space and the resulting trajectories are guaranteed to be globally optimal up to the resolution of discretization. However, hypervolume increases exponentially as a function of dimensionality and so does number of grids required to attain spatial density in it. As a result, these algorithms become infeasible in high dimensions. This phenomena is often reffered to as the curse of dimensionality.

Many variations of DP have been proposed to avoid the curse of dimensionality. Focus of non-parametric approaches is on how to reduce number of data points at which to represent a value while representing the state space well. State Increment Dynamic Programming [1] achieved this by increasing the size of each grid and Adaptive grid scheme [2], [3] adapted size of the grids. Atkeson has shown sampling of states [7] and actions [8] can both lead to more efficient optimization. Parametric approaches parameterize values over region of hypervolume [9], [10]. These variants of DP have focused on how to better represent the state space with less representational resources, but still suffer from the curse of dimensionality because the growth of required representational resources is still exponential. Local trajectory optimization technique such as Differential Dynamic Programming (DDP)[4], [5] have been used to speed up the convergence of the value function [6]. [6] also showed how to generate locally optimal trajectories without nominal trajectories based on DDP and used them as key trajectories to approximate value functions around them.

A classic approach to optimal control problem is to solve boundary value problem of differential equations [11], [12]. These approaches require a nominal trajectory or initial guess on which the resulting trajectory is highly dependent. These approaches do not suffer from the curse of dimensionality but generating nominal trajectories which lead to a good trajectory is not trivial for complex tasks. A* and randomized search algorithm such as RRT's [13] can generate reasonable trajectories without severely suffering from the curse of dimensionality given good heuristics. However, obtaining good heuristics requires extra work such as hand tuning or training examaples, and without good heuristics A* would suffer severely from the curse of dimensionality and RRT's would not be able to provide reasonable trajectories. As a result, these approaches require domain specific knowledge to generate reasonable trajectories.

In this paper, we present an approach for generating locally optimal trajectories without nominal trajectories or heurestics. Our approach does not cover any hypervolume of state space and only requires polynomial memory and time. The objective of our approach is to generate a variety of locally optimal trajectories to increase the chance of obtaining a reasonable trajectory.

II. APPROACH

A key idea of our approach is to generate locally optimal trajectories without nominal trajectories nor covering a volume of state space [6]. With random choice of initial conditions, we explore a variety of locally optimal trajectories. The wider variety of locally optimal trajectories we generate, the better chance there is to get a reasonable trajectory. The recursive procedure for extending locally optimal trajectory backwards in time is explained in section II-A. This procedure governs the system's behavior and trajectory is determined given the initial conditions. A method to find initial conditions which satisfy the endpoint constraints is then presented in II-A.1. Finally, complexity of the approach is discussed in II-A.2.

System dynamics is $\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ and one time cost is $L(\mathbf{x}, \mathbf{u})$ in discrete time. Value function $V(\mathbf{x})$ is the total cost to the goal and is equivalent to $\sum_{t=0}^{\infty} L(\mathbf{x}, \mathbf{u}^{opt})$. For the approach to work, it is assumed that dynamics and cost function are continuous to second order.

A. Extending trajectories backward

Optimal action $\mathbf{u}^{opt}$ can be generally expressed by Bellman equation

$$\mathbf{u}^{opt} = \mathbf{arg}_u \mathbf{min} Q(\mathbf{x}, \mathbf{u}) \tag{1}$$

where $Q(\mathbf{x}, \mathbf{u}) = L(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}, \mathbf{u}))$. The Bellman equation implies optimal action cannot be determined without knowing V. Much effort of DP goes into obtaining V which requires covering a hypervolume and thus suffers from the curse of dimensionality. We start planning from near the goal state where we can obtain a local value function model relatively easy, and maintain a second order local model of it as the trajectory is extended backwards [4], [5]. By planning backwards and maintaining a value function model, we can generate locally optimal trajectories without having to cover hypervolume. An initial value function model at $\mathbf{x}^{goal}$ and $\mathbf{x}(t^f)$ needs be chosen, and a locally optimal trajectory leading to $\mathbf{x}(t^f)$ forwards in time is extended backwards in time recursively. $\mathbf{x}(t^f)$ is a state near the goal and represents how a trajectory reaches the goal. This approach of planning backwards draws inspiration from DDP [4], [5] which refines a nominal trajectory by propagating local value function model backwards in time along it and performing gradient descent based on it forwards in time.

1) Findinging the optimal action: Using quadratic approximations of the various functions, the optimal action $\mathbf{u}^{opt}$ satisfies $Q_u(\mathbf{x}, \mathbf{u}^{opt})$[1] $= 0$. Thus given a local model of $V(\mathbf{x}(t+1))$, $\mathbf{u}^{opt}(t)$ can be determined iteratively by

$$\mathbf{u}^{i+1} = \mathbf{u}^i - \epsilon Q_u(\mathbf{x}, \mathbf{u}^i) \tag{2}$$

where Q_u can be given by (9) and $0 \leq \epsilon \leq 1$. The iteration is terminated when $\|Q_u\|^2$ is sufficiently small.

2) Finding the predecessor state: Since we are extending trajectories backwards in time, we would like to find the state whose optimal action leads to the current state in one timestep. If the start of the trajectory is at state $\mathbf{x}^{cur}$ at time $t+1$, predecessor state $\mathbf{x}^{pre}$ must satisfy $\mathbf{x}^{cur}$ = $\mathbf{f}(\mathbf{x}^{pre}$,$\mathbf{u}^{opt})$. This can be found iteratively

$$\delta\mathbf{x}^i(t) = \mathbf{x}^{cur} - \mathbf{f}(\mathbf{x}^i(t), \mathbf{u}^{opt}(t)) \tag{3}$$

$$\mathbf{x}^{i+1}(t) = \mathbf{x}^i(t) + \epsilon\delta\mathbf{x}^i(t) \tag{4}$$

where $\mathbf{u}^{opt}$(t) can be obtained by (2). The iteration tis terminated when $\left\|\delta\mathbf{x}^i(t)\right\|^2$ is sufficiently small. A unique predecessor exists if $\mathbf{f}_x - \mathbf{f}_u Q_{uu}^{-1} Q_{ux} \neq 0$ for second order functions of $\mathbf{f}$ but cannot be guaranteed for general cases. In general, there does not have to be a unique parent for this approach to work since whichever parent state is chosen the trajectory is still locally optimal.

3) Updating and Propagating V and Q models backwards in time: Once $\mathbf{x}(t)$ is found and trajectory is extended backwards by one timestep, quadratic models of the value function and Q function can be updated and propagated to $\mathbf{x}(t)$ as follows [4], [5].

$$V(t) = V(t+1) - Q_u(t)Q_{uu}^{-1}(t)Q_u(t) \tag{5}$$

$$V_x(t) = Q_x(t) - Q_u(t)Q_{uu}^{-1}(t)Q_{ux}(t) \tag{6}$$

$$V_{xx}(t) = Q_{xx}(t) - Q_{xu}(t)Q_{uu}^{-1}(t)Q_{ux}(t) \tag{7}$$

$$Q_x(t) = V_x(t+1)\mathbf{f}_x + L_x(t) \tag{8}$$

$$Q_u(t) = V_x(t+1)\mathbf{f}_u + L_u(t) \tag{9}$$

$$Q_{xx}(t) = \mathbf{f}_x^T V_{xx}(t+1)\mathbf{f}_x + V_x(t+1)\mathbf{f}_{xx} + L_{xx}(t) \tag{10}$$

$$Q_{xu}(t) = \mathbf{f}_x^T V_{xx}(t+1)\mathbf{f}_u + V_x(t+1)\mathbf{f}_{xu} + L_{xu}(t) \tag{11}$$

$$Q_{uu}(t) = \mathbf{f}_u^T V_{xx}(t+1)\mathbf{f}_u + V_x(t+1)\mathbf{f}_{uu} + L_{uu}(t) \tag{12}$$

B. Satisfying a Start-End Boundary Condition

The previous section showed the recursive procedure for extending a locally optimal trajectry backwards in time. [6] proposed to use these trajectories as key trajectories to represent the value function and did not aim at satisfying the start-end boundary condition with these trajectories. Here, we reduce the problem to a two point boundary value problem (TPBVP) whose system is gorvened by $\mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}^{opt}(t))$, and boundary conditions are $\mathbf{x}(t^0) = \mathbf{x}^{start}$ and $\mathbf{x}(t^f) = \mathbf{x}^{goal}$. Section II-B.1 and II-B.2 describe the two methods we use to satisfy the TPBVP and II-B.3 introduces another variable which needs be selected.

1) Multiple Shooting Method: Shooting methods reduce a TPBVP to an initial value problem [11], [12]. Trajectories are generated from near the goal state in our approach, and thus correction to $\mathbf{x}(t^f)$ should be made such that error at the start boundary is decreased. Once a trajectory is generated from $\mathbf{x}(t^f)$ which does not satisfy the start-end boundary condition, correction to the initial condition is found as follows.

$$\mathbf{K}(t) = Q_{uu}^{-1}(t)Q_{ux}(t) \tag{13}$$

$$\mathbf{U}(t+1) = (\mathbf{f}_x(t) - \mathbf{f}_u(t)\mathbf{K}(t))\mathbf{U}(t) \tag{14}$$

$\mathbf{f}_x$ and $\mathbf{f}_u$ approximate the dynamics, and feedback gain matrix $\mathbf{K}$ approximates the policy [4], [5] in the neighborhood of current trajectory. By integrating $\mathbf{U}(t)$ from time t^0 to t^f with $\mathbf{U}(t^0)$ being an identity matrix, $\mathbf{U}(t^f)$ is found. Then correction to the initial condition $\mathbf{x}(t^f)$ can be found by $\delta\mathbf{x}(t^f) = \mathbf{U}(t^f)\delta\mathbf{x}(t^0)$ where $\delta\mathbf{x}(t^0)$ is the correction that you would like to make at the start boundary. Once $\delta\mathbf{x}(t^f)$ is found, a new trajectory is generated from $\mathbf{x}^{i+1}(t^f)$ = $\mathbf{x}^i(t^f)$ + $\delta\mathbf{x}(t^f)$ by following the procedures described in section II-A.

[1] Q_u is a derivative of Q with respect to u.

2) Relaxation Method: Relaxation method in our case satisfies the boundary condition by relaxing the local optimality of the trajectory. A new trajectory is generated by integrating the dynamics forward in time starting at $\mathbf{x}^{i+1}(t^0) = \mathbf{x}^{start}$ using the feedback gain of the original trajectory obtained by (13).

$$\delta\mathbf{u}(t) = -\mathbf{K}(t)(\mathbf{x}^{i+1}(t) - \mathbf{x}^{i}(t)) \tag{15}$$

$$\mathbf{x}^{i+1}(t+1) = \mathbf{f}(\mathbf{x}^{i+1}(t), \mathbf{u}^{i}(t) + \delta\mathbf{u}(t)) \tag{16}$$

The new trajectory is not locally optimal but can be refined using local trajectory optimization techniques such as DDP.

3) Floating Time: Unless terminal time t^f is specified, it has to be determined when solving boundary value problem using either method. Given a trajectory, it has to be determined at which point on the trajectory the boundary condition should be satisfied. Together with $\mathbf{x}(t^f)$, t^f is responsible for determining which locally optimal trajectory is generated. How t^f is chosen for our specific application is disccused in later section.

C. Complexity

Finding a locally optimal policy requires $\Theta(d^3)$ cost dominated by computing the second order Taylor expansion of **f**. The process of finding predecessor state requires finding an optimal policy for each candidate predecessor state. At each iteration of (3)-(4), the error, distance between the current state and candidate predessor state's optimal child state, reduces linearly as $\left\|\delta\mathbf{x}^{i+1}(t)\right\|^2 \leq \frac{1}{C}\left\|\delta\mathbf{x}^{i}(t)\right\|^2$ where $C > 1$, thus the number of iterations required to satisfy error tolerance of *e* is $\log_C \frac{\left\|\delta\mathbf{x}^{0}(t)\right\|^2}{e}$. Expected initial error $\left\|\delta\mathbf{x}^{0}(t)\right\|^2$ is proportional to *d* and thus complexity of finding predecessor state is $\Theta(d^3 \cdot log\ d)$. For a single trajectory without boundary conditions, computational cost is $\Theta(T \cdot d^3 \cdot log\ d)$ where *T* is the total number of time steps of the trajectory. For a trajectory with randomly selected $\mathbf{x}(t^f)$, the expected squared distance from the start state is proportional to *d*, and only constant convergence is possible with either shooting or relaxation method when the trajectory is not sufficiently close to it. Thus, the time complexity of otbaining a trajectory satisfying start-end boundary condition is $\Theta(T \cdot d^4 \cdot log\ d)$.

Representational cost per state $\Theta(d^3)$ is dominated by the second order Taylor series of **f**, and $\Theta(T \cdot d^3)$ per trajectory of length *T*. Since only previous and current trajectories need be stored, total representational requirement of the approach remains $\Theta(T \cdot d^3)$.

III. PENDULUM SWING-UP TASK

The approach is tested on a pendulum swing-up task with 1,2 and 3 links. The task is to swing up a pendulum which is initially hanging down to point straight up which is an unstable equilibrium position. The pendulum has total length of $1m$ and total mass of $1kg$ which is distributed uniformly. Each link is $\frac{1}{n}m$ long, $0.1m$ wide and weighs $\frac{1}{n}kg$ where *n* is the number of links. Joints are frictionless and only external force acting on the system is gravity of $9.81\frac{m}{s^2}$. Sample time of 0.01 second is used and total length of the trajectory is limited to less than or equal to 7 seconds. Each joint is limited to $-2\pi < \theta < 2\pi$, and there is no limit on torque or joint speed. Optimization criteria penalizes weighted sum of squared distance from the goal in joint coordinates and squared torques, but angular velocities are not penalized.

$$L(x,u) = \delta t \cdot \sum_{i=1}^{n}(0.1 \cdot (\frac{\theta_i - \theta_i^{goal}}{rad})^2 + (\frac{\mathbf{u}_i}{rad/sec})^2) \tag{17}$$

where θ_i and $\mathbf{u}_i$ are angle and applied torque of ith joint respectively.

IV. RESULTS

The second order value function model at $\mathbf{x}^{goal}$ was found by solving discrete-time Riccati equation at $\mathbf{x}^{goal}$.

$$f_x{}^T V_{xx} f_x - V_{xx} - (f_x{}^T V_{xx} f_u)(f_u{}^T V_{xx} f_u + L_{uu})^{-1}(f_u{}^T V_{xx} f_u) + L_{xx} = 0 \tag{18}$$

V_x is initialized to be zero because both dynamics and cost function are symmetric about $\mathbf{x}^{goal}$. $\mathbf{x}(t^f)$ is initially sampled uniformly on the hypersphere around $\mathbf{x}^{goal}$ whos squared radius $\sum_{i=1}^{n}(\frac{(\theta_i - \theta_i^{goal})^2}{rad} + \frac{\dot{\theta}_i^2}{rad/sec})$ is equal to 0.001. t^f disscussed in section II-B.3 is taken to be the points on initial trajectory which are local minima in terms of distance to $\mathbf{x}^{start}$. In addition to the approach presented in this paper, we have also implemented standard DP using a resolution 100 for each dimension for 1- and 2-link pendulum swing-up. Trajectories generated by standard DP are refined using DDP later. For 3 links, we could not afford 8 terabytes of memory required to represent a value of type "double" at 100^6 grid locations. Experiments were performed using Matlab on a 3GHz Pentium IV with 1GB of RAM running Linux.

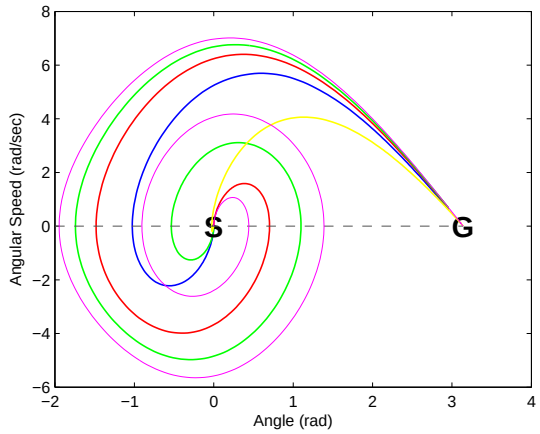

Fig. 1. Locally optimal trajectories for 1-link pendulum swing-up in phase space.

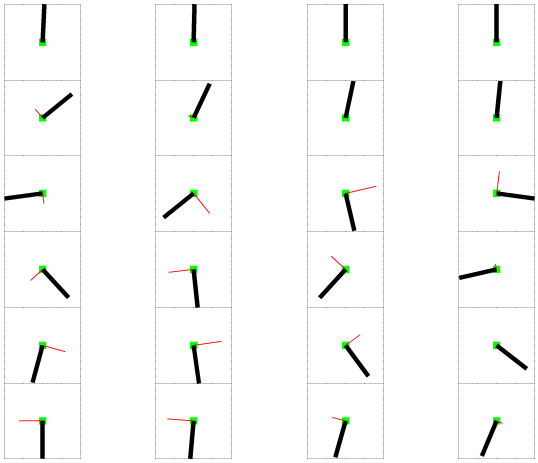

Fig. 2. Best trajectory for 1-link pendulum at every 0.2 sec. The line at the joint drawn perpendicular to the link is the relative magnitude and direction of the applied torque. Time flows from left to right, bottom to top.

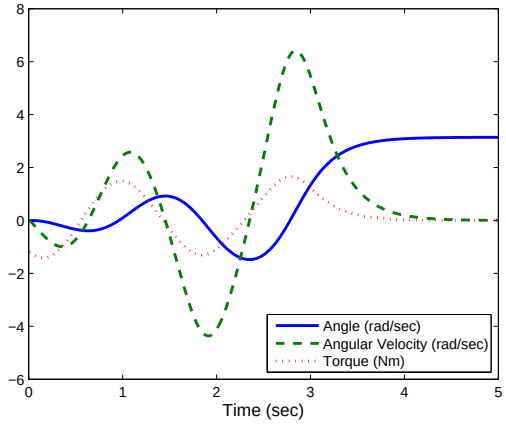

Fig. 3. Best trajectory for 1-link pendulum.

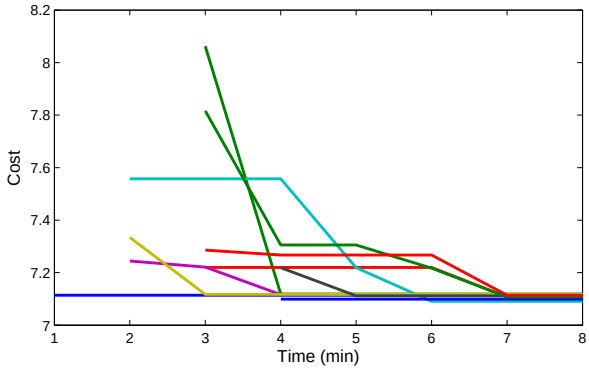

Fig. 4. 10 sample runs of our approach for 1-link pendulum swing-up.

A. 1 link

Fig. 1 shows locally optimal trajectories generated by our approach who approach the goal from the right. What makes these trajectories different from one another is the initial choice of $\mathbf{x}(t^f)$ and $\mathbf{x}(t^0)$ disscussed earlier. Best trajectory by our approach shown in Fig. 2,3 has total score of 7.09. Fig. 4 shows 10 runs of our approach with different random number generator seeds for which we found costs of 7.1098 ± 0.0088 after running 8 minutes. For standard DP approach, [0, 2π] rad was covered for angle and [-10, 10] rad/sec for angular velocity and uniformly discretized into 100 grids each. For action space, torque of [-5, 5] Nm was descritized uniformly with 11 resolutions. The solution gave total score of 7.31 after 10 minutes.

B. 2 links

The best trajectory we found has total cost of 5.09 and is shown in the Fig. 5, 9, 11 and 13. Fig. 6 shows 10 different runs of our approach for which we found costs of 6.7655 ± 0.8856 after running 30 minutes. For standard DP approach, [0, 2π] rad was covered for angles and [-40, 40] rad/sec for angular velocities and uniformly discretized into 100 grids each, and [-5, 5] Nm of action space were descritized to 11 resolutions for each joint. The solution gave total score of 5.7 after 18 hours. Much bigger regions of angular velocity spaces are covered compared to that of 1-link pendulum, and thus the uncertainty is higher, which may have led to the poor score compared to the best trajectory by our approach.

C. 3 links

Our best trajectory for 3-link pendulum has total cost of 4.61 shown in Fig. 7, 10, 12 and 14. We tried 10 different runs and found costs of 5.0000 ± 0.3030 after 70 minutes. No standard DP solution is available for this system.

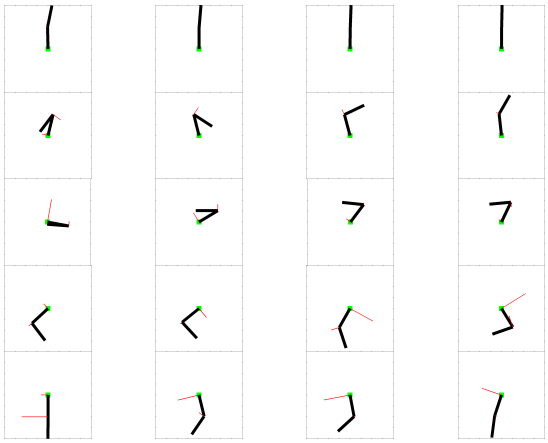

Fig. 5. Best trajectory for 2-link pendulum at every 0.1 sec.

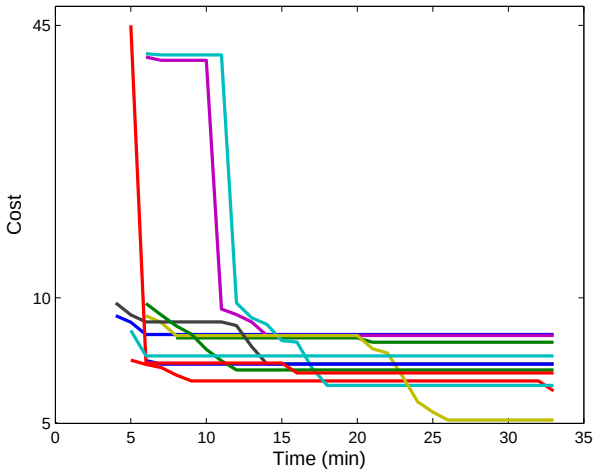

Fig. 6. 10 sample runs of our approach for 2-link pendulum swing-up. Note the log scale on the vertical axis.

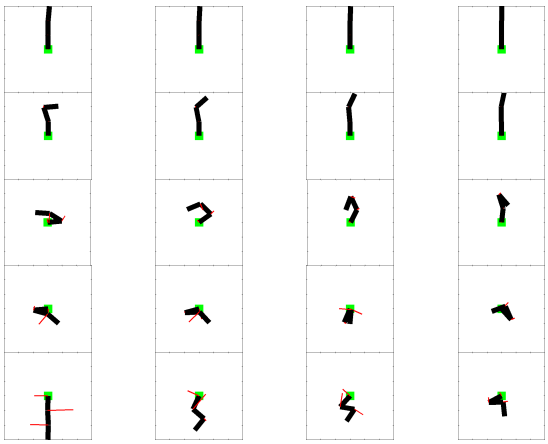

Fig. 7. Best trajectory for 3-link pendulum at every 0.02 sec.

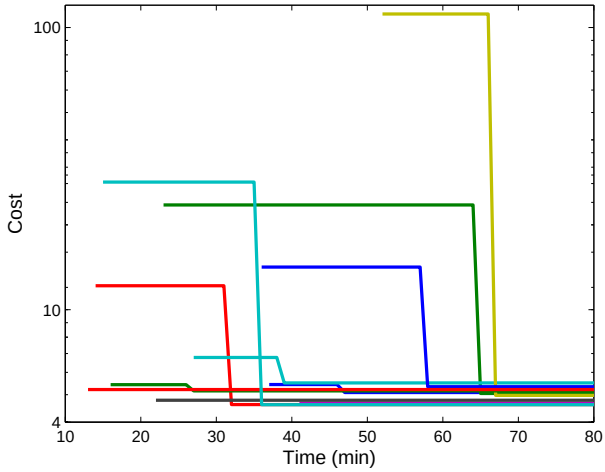

Fig. 8. 10 sample runs of our approach for 3-link pendulum swing-up. Note the log scale on the vertical axis.

V. CONCLUSIONS

The contribution of this work is the demonstration that reasonable trajectories can be obtained without covering hypervolume. Even though the curse of dimensionality is not avoided because our approach cannot guarantee any global optimality, we have shown experimentally that trajectories as good as standard DP can be obtained using our approach using pendulum swing-up task with 1 and 2 links. We have also shown experimentally that our approach outperforms standard DP in terms of time complexity. Furthermore, we have demonstrated that locally optimal trajectory can be generated for 3-link pendulum for which standard DP approach is currently infeasible.

VI. FUTURE WORK

Future work would address more challenging problems such as pendulum swing-up with more links or under existence of obstacles in state space. Comparison of the trajectories obtained by our approach to those obtained by other approaches should be given to verify the optimality. Another direction for the work is high level framework which reasons or learns what kind of a locally optimal trajectory would be more optimal than others so that better locally optimal trajectories can be generated more efficiently than random sampling.

VII. ACKNOWLEDGMENTS

The author thanks Christopher G. Atkeson for the invaluable comments and discussions.

REFERENCES

[1] Robert E. Larson. *State Increment Dynamic Programming*. American Elsener Publishing Company, New York, 1968.

[2] R. Munos, A. Moore. *Variable resolution discretization in optimal control*, Machine Learning 49, pp. 291-323. 2002.

[3] L. Grune, W. Semmler. *Using dynamic programming with adaptive grid scheme for optimal control problems in economics*, Journal of Economic Dynamics and Control, Elsevier, vol. 28(12), pp. 2427-2456, 2004.

[4] P. Dyer & S.R. McReynolds. *The Computation and Theory of Optimal Control*. Academic Press, New York, NY, 1970.

[5] D.H. Jacobson & D.Q. Mayne. *Differential Dynamic Programming.* Elsevier, New York, NY, 1970.

[6] C. Atkeson. *Using Local Trajectory Optimizers To Speed Up Global Optimization In Dynamic Programming*, In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, Advances in Neural Information Processing Systems, volume 6, pages 663-670. Morgan Kaufmann Publishers, Inc., 1994.

[7] C. Atkeson. *Randomly Sampling States In Dynamic Programming*, Neural Information Processing Systems, 2007.

[8] C. Atkeson. *Randomly Sampling Actions In Dynamic Programming*, Symposium on Approximate Dynamic Programming and Reinforcement Learning, 2007.

[9] J.A. Bagnell, J.Schneider. *Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods*, International COnference on Robotics and Automation, 2001.

[10] H. Benitez-Silva, J. Rust, G. Hitsch, G. Pauletto, G. Hall. *A Comparison Of Discrete And Parametric Methods For Continuous-State Dynamic Programming Problems*, No 24, Computing in Economics and Finace, 2000.

[11] G. Fraser-Andrews. *A multiple-shooting technique for optimal control*, Vol. 102 Issue 2, Journal of Optimization Theory and Applications, 1999.

[12] D. D. Morrison, J. D. Riley, J. F. Zancanaro. *Multiple Shooting Method For Two-Point Boundary Value Problem*, Communications of the ACM, pp.613-614, 1962.

[13] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, New York, NY, 2006.

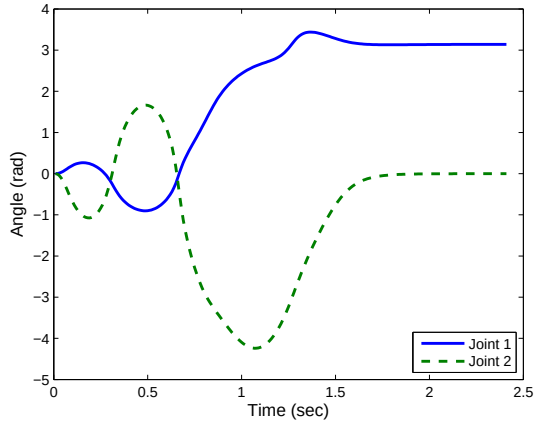

Fig. 9. Best trajectory for 2-link pendulum.

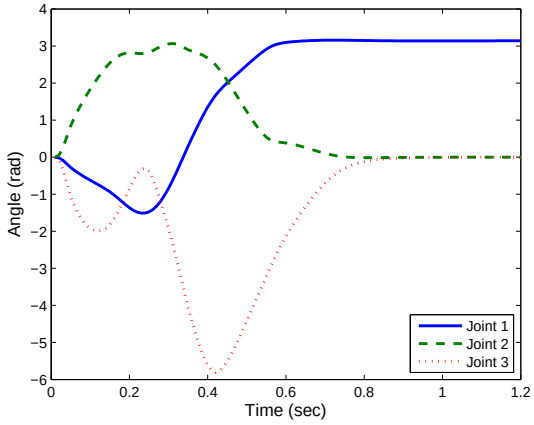

Fig. 10. Best trajectory for 3-link pendulum.

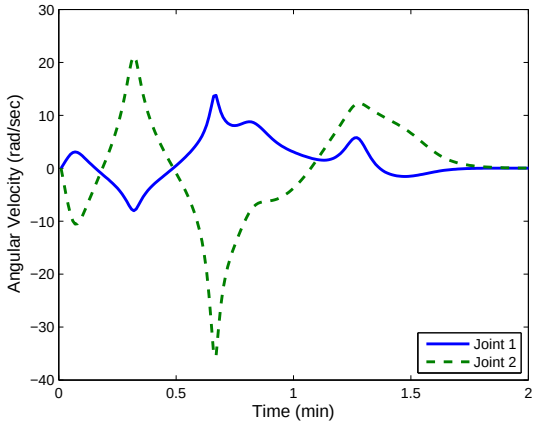

Fig. 11. Best trajectory for 2-link pendulum.

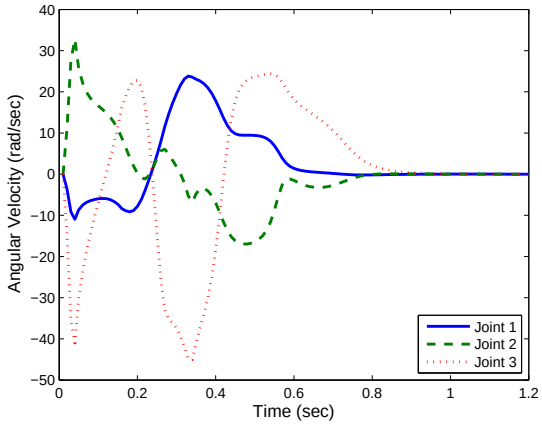

Fig. 12. Best trajectory for 3-link pendulum.

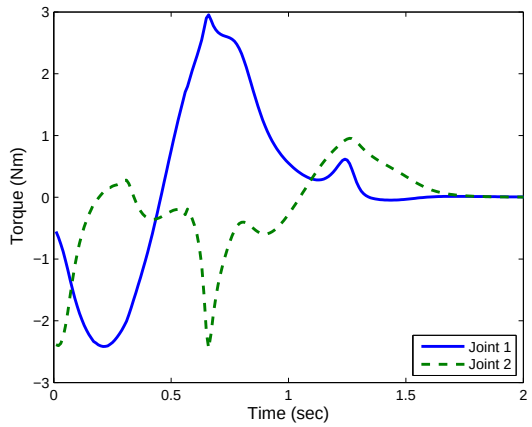

Fig. 13. Best trajectory for 2-link pendulum.

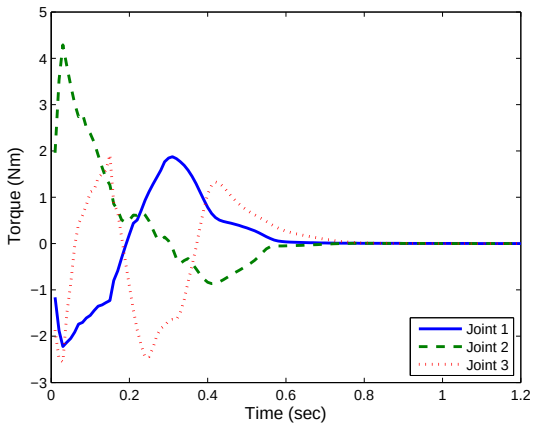

Fig. 14. Best trajectory for 3-link pendulum.