Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

Efficient Reinforcement Learning: Model-based Acrobot Control

Gary Boone
College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280
gboone@cc.gatech.edu, www.cc.gatech.edu/~gboone

Abstract—**Several methods have been proposed in the reinforcement learning literature for learning optimal policies for sequential decision tasks. Q-learning is a model-free algorithm that has recently been applied to the Acrobot, a two-link arm with a single actuator at the elbow that learns to swing its free endpoint above a target height. However, applying Q-learning to a real Acrobot may be impractical due to the large number of required movements of the real robot as the controller learns. This paper explores the planning speed and data efficiency of explicitly learning models, as well as using heuristic knowledge to aid the search for solutions and reduce the amount of data required from the real robot.**

I. Introduction

REINFORCEMENT learning algorithms learn to perform sequential decision tasks without explicit instruction by optimizing a criterion as they perform the task. These algorithms are naturally applied to robot control problems in which optimal policies are not available through analytic means. The Acrobot, an underactuated two-link robot arm, is a nonlinear system with discrete or continuous actions that can be used for several tasks. Q-learning techniques have been used successfully to learn control policies for a discrete action, minimum-time swing-up Acrobot task [1]. We have reimplemented and confirmed these results, but wish to consider the implications of these techniques for physical robots.

Because moving the physical robot during learning requires real time, we are motivated to seek algorithms which are *data efficient*, requiring as little data as possible from the system to be controlled. Such algorithms maximize the utility of the data that is collected, perhaps even storing all of the data collected over the system's lifetime. Given the increasing speed and storage capacity of computers, data from the real system can become the limiting resource for rapid learning.

One goal of learning algorithms is *generality*: wide applicability and minimal use of domain knowledge. However, the need for data efficiency may override the need for generality. In many robotics problems, the use of domain knowledge allows us to find optimal control policies with a minimum of exploration using the physical system.

Reinforcement learning emphasizes learning in which an agent interacts with an unknown environment, learning to perform an unknown task optimally. However, in many situations it is useful to distinguish between these two related tasks. Learning about the system to be controlled entails the construction of a model of the effects of the agent's actions. Learning to behave optimally means learning a policy that provides the agent with the best action for a given state. There are several learning methods that do not attempt to model the system. These *direct* methods learn optimal policies directly from the reinforcements. However, for problems such as the Acrobot in which the fast and accurate models can be constructed, the need for data efficiency motivates the use of modeling because models allow mental simulation [2]. By creating a model of the real system, the learning agent is provided a means of exploring to improve the policy without driving the real system. Further, modeling allows the system to more effectively choose which data to request from the system. Modeling aids both *exploitation* and *exploration*.

This paper considers the use of domain knowledge, simple heuristics, and modeling to considerably improve the data efficiency of learning to control the Acrobot. After describing the Acrobot and the Q-learning system we replicated [1], we consider several heuristic approaches to finding optimal trajectories without models. Then we examine the utility of modeling and show show our heuristics can be used with models to learn offline. Finally, we compare our modeling approaches to two direct learning methods, Q-Learning with tables and with CMACs.

II. The Acrobot

The Acrobot is a nonlinear dynamical system composed of a two-joint arm actuated only at the elbow joint [1]. One learning task for this system is to raise the endpoint to a preset height in a minimal number of steps. A more difficult task is to swing up to a vertical position and remain upright. The torques applied at

0-7803-3612-7-4/97 $5.00 © 1997 IEEE

the elbow may be continuous or discrete. This paper considers the task of swinging to a target height with three discrete torques: +1 Nm, 0, and -1 Nm. The Acrobot and its swingup tasks have been studied in the control literature; several techniques for designing controllers have been proposed [3], [4], [5]. This paper does not focus on finding controllers, but instead considers the utility of domain knowledge and modeling to improve the performance of reinforcement learning agents.

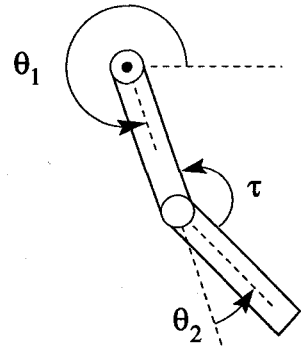

Figure 1. The Acrobot. The Acrobot is a two-link robot arm that swings freely at joint 1 and is actuated at joint 2. The state variables are $\theta_1, \dot{\theta}_1, \theta_2$, and $\dot{\theta}_2$.

Problems which minimize time under the constraint of finite control magnitudes are often called *minimum-time* problems and have been studied extensively in the control literature. It is known that in many cases, minimum-time problems are optimized by operating the controls at their extremes [6]. Finding the optimal control for a minimum time problem reduces to finding a surface in state space; the controller switches maximal actions when the system trajectory crosses the surface. To solve the minimum-time Acrobot problem, a learning system must represent and optimize the switching surface. Two important issues are choosing a representation and finding the optimal location of the switching surface. Below we describe several ways of representing and finding these surfaces, including linear and non-linear models, tables, and function approximators such as CMACs. These representation methods can also be used to model the Acrobot dynamics while learning the optimal switching policy.

III. Learning Policies without Models

Because the Acrobot controller can switch actions at each time step, even short trajectories have a vast number of possible switching plans. For example, a trajectory of 50 steps has approximately 10^{15} possible switching sequences. To find even moderate-quality solutions, the learning system must severely prune its search of this space.

A. Q-Learning and Sarsa

Q-learning is a learning algorithm for sequential decision tasks that learns the value of applying an action in any state [7]. The value of an action is defined as the cost of the action plus the maximal future reward, assuming the optimal policy is followed after the action is taken. Because Q-learning learns these values for every action and every state, the optimal policy can be greedy, taking the applicable action that has the highest value.

Q-learning is a *direct* learning method because it learns a policy directly without learning a model of the system. However, to learn the values of every action, it must visit each state and apply each action often enough to learn the effects of the actions. If the system is stochastic, Q-learning requires that every state be visited infinitely often. Even for deterministic systems, such as our formulation of the Acrobot, Q-learning requires enough data to correctly estimate the effect of any action.

Our implementation of Q learning required three four-dimensional tables, one for each Acrobot action. Each table had a resolution of $6 \times 6 \times 6 \times 6$. The Q/Table approach learned trajectories as short as 116 steps, which required 401,593 Acrobot steps, as shown in Table I.

Better results can be obtained using a eligibility traces and a function approximator to provide generalization. Our implementation follows Sutton[1]. Eligibility traces explicitly retain knowledge of previously visited states, giving them some portion of subsequent discounted rewards. Sarsa modifies Q-learning by using the Q value of the action actually taken rather than the best available Q for the update [8], [1]. However, if a greedy policy is followed, Sarsa is identical to standard Q-learning. As in [1], we used a greedy policy. This approach required 94,830 Acrobot training steps and found a solution of length 76. These results replicate [1].

B. Heuristic Approaches

Using Q-learning allows the Acrobot controller to learn policies from only a reinforcement signal. There is more information available to the learning system; avoiding the applicable domain knowledge makes the learning task unnecessarily difficult. Further, it requires the learner to generate more data with the real robot than systems which can effectively use *a priori* or acquired domain knowledge.

How can domain knowledge be used to control the Acrobot? This section suggests fixed policies and search methods for finding trajectories without using models. In Section IV, we will use these methods to

plan using models rather than actual Acrobot movements.

B.1 A Simple, Fixed Policy

To swing the Acrobot above the goal, the controller must gradually pump energy into the Acrobot's motion. Observing this, we designed a system that explicitly used an energy "pumping" heuristic to choose actions. Using a fixed policy given by

$$u(\mathbf{x}) = -sgn(\dot{\theta}_1),$$

where $\mathbf{x}$ is the system state, is equivalent to defining the switching surface as a plane, $\dot{\theta}_1 = 0$. This simple policy generated trajectories of length 96 with no learning. The policy is clearly sub-optimal because it does not try to raise the Acrobot's endpoint above the goal height directly once sufficient energy has been pumped into the system. However, the fixed policy is better than the trajectories found by table-based Q-learning.

B.2 Heuristic Search Methods

The same pumping heuristic can be used to guide a search by looking for the switching points that maximize individual swings. The heuristic search algorithm considered each swing of the upper link as a new search. For each segment, it maximized the angle of the upper link when the link stopped its upward swing. Specifically, for each segment between zero crossings of $\dot{\theta}_1$, the system found the actions that maximized $|\theta_1|_{\dot{\theta}_1=0}$.

We simulated the Acrobot with the parameters given in [1], also using Euler integration. Using this method, we were able to find a trajectory that could reach the goal in 63 steps (Figure 2). Although proving its optimality would require an exhaustive search, this is the shortest trajectory that we have seen. The search process evaluated a total of 288 states. Note that this search method finds a switching policy that does not include the no-torque action.

B.3 A*-based Search Methods

Dynamic programming [9] is the basis of many reinforcement learning systems. By flooding solutions backwards from the goal, dynamic programming efficiently searches the state space by finding all optimal n-step paths that lead to the goal. Once n is large enough to include the start state, the optimal trajectory is known. Reinforcement algorithms such as Q-Learning are forward, incremental versions of DP; they update estimates of the value function as they try actions in the environment. Like DP, values converge near the goal first and flood backwards.

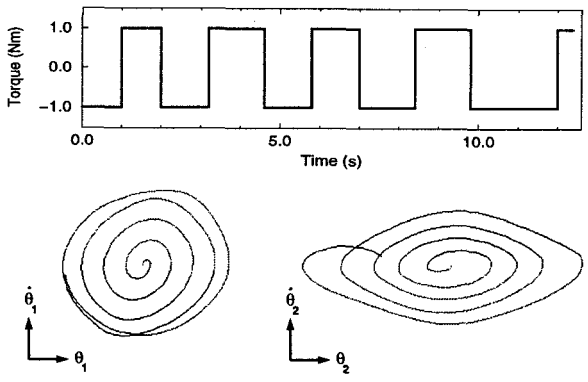

Figure 2. **Best Observed Acrobot Solution.** This trajectory required only 63 steps. The phase plots show two-dimensional projections of the link angular velocities (vertical) and angles (horizontal). The left and right phase plots correspond to the first and second links, respectively. The trajectory spirals out from the center until enough energy has been pumped into the system that the second link can be extended to the goal height.

A* is another common search technique [10]. It uses estimates of the distance to the goal to search efficiently. The technique is guaranteed to converge provided that the estimates are underestimates of the true cost. Applying more domain knowledge, we chose

$$C_{est} = \frac{y_{\text{goal}} - y_{\text{tip}}}{\dot{y}_{\text{tip}}^{\max}}$$

as the estimate of the steps remaining to reach the goal. y_{tip} is the vertical height of the endpoint of the Acrobot. $\dot{y}_{\text{tip}}^{\max}$ is the endpoint's maximum vertical speed.

In many problems, there are many actions available at any state, leading to intractable search trees if trajectories are constructed forward from the start state. Dynamic programming is efficient because it confines its search to only those trajectories capable of reaching the goal. A* is efficient because it continues those trajectories that appear to have the smallest total cost. For systems with continuous actions, there are an infinite number of actions. However, our version of the Acrobot has only three possible actions at each time step, making forward searches practical if some pruning is used.

We pruned the search tree by dividing the space with a grid and allowing only one trajectory to enter each cell. Starting from the initial condition, the actions are applied until a trajectory leaves a cell. As each new state is calculated, it is stored on a queue of trajectory endpoints. However, once a trajectory enters a cell, any other trajectory with a larger accumulated cost that enters the cell is discarded, as shown in Fig-

ure 3. By using a priority queue, we always extend the minimum-length trajectories first. The search is terminated when the goal is reached. We also found that algorithm performance could be improved by continuing each action for 3 control time steps.

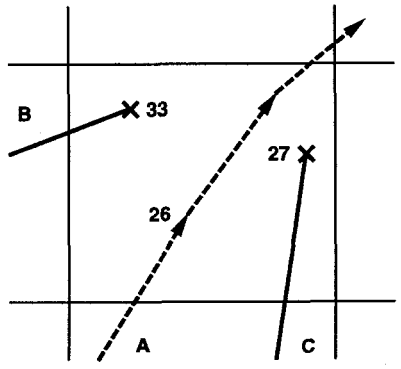

Figure 3. **Trajectory Pruning.** During the search, the state space is covered with a uniform grid which is used to prune nearby trajectories. For example, trajectory A enters the grid cell after 26 steps; later trajectories that enter the cell with larger step costs, such as B and C, are immediately deleted from the search.

Using this search technique on a perfect model of the Acrobot, we found trajectories of length 68. The resolution of the pruning grid was $23 \times 23 \times 23 \times 23$. For additional discussion of direct search methods for time-optimal Acrobot trajectories, see our companion article [11].

IV. Learning the System Model

Like Q-learning, the above methods allow us to find good trajectories for the Acrobot minimum-time task. Further improvements in data efficiency can be achieved by minimizing the use of the physical system during planning. Learning a model allows the agent to explore the effects of actions on a fast, simulated time scale rather than a physical time scale. Unfortunately, Q-learning is typically used to avoid model-building because it can directly learn policies from the reinforcement signal. In the following sections, we suggest modeling methods and show how modeling reduces the amount of data required for planning. The use of domain knowledge can further increase the effectiveness of modeling by reducing the amount of data required for model building.

A. Model Learning with Domain Knowledge

Because we know the structure of the system, the most natural model is given by the analytically derived equations of motion. However, for a real robot, we may not know the precise physical parameters, such as link lengths and masses. Further, the equations may not include phenomena such as friction or actuator errors. We can define a parameterized linear model and use regression on the actual data to determine the correct parameters. To construct such a model, the equations of motion are written as a sum of nonlinear terms. The constants that weight these terms become the unknown regression variables.

To generate training data, the Acrobot is given a random trajectory. The data is observed and used in a linear regression. Because the simulated Acrobot does not have noise or unmodeled effects, only enough data is required to make the regression matrix nonsingular. Our system executed 20 random steps, then used regression to construct a model with which it was able to find a 68 step solution. A real Acrobot would require more data; the values reported below represent an idealized case and are intended to show one extreme possible when good models are available to the learning agent.

B. Model Learning with Less Domain Knowledge

We also wanted to explore the use of more general function approximators. Following Sutton [1], we used CMACs, which consist of overlapping grids [12]. Instead of representing a Q function, we used the CMACs to learn a model of the system. The Cerebellar Model Articulation Controller is a function approximator based on linear sums of the contents of offset grids. That is, a set of grids is overlapped, each randomly offset from the first grid. A query point will be contained in one cell of each grid. To answer the query, the contents of the relevant cell from each grid are retrieved and summed. In our planners, the CMACs are used to model the system accelerations for any state and applied torque. Integrating these accelerations determines the subsequent state. There were two joint accelerations to compute for three possible torques applied by the Acrobot controller, requiring a total of six CMACs.

We applied both our heuristic search and A*-based planners to the CMAC model. After the planners create a desired trajectory, the true Acrobot is given the indicated commands. If the actual path deviates from the planned path by more than a preset distance in state space, the planner is reinvoked. The data collected from the actual movement is used to update the model before creating a new plan. For the heuristic search planner, we trained on all the data seen by the robot during training. For the A*-based planner, we trained only on the data from the last trajectory. Both planners made five training passes through the data each time new data was collected. Note that by constantly replanning, the planner guides the search in the most promising direction given the current model.

Model?	*Method*	*Shortest Trajectory Found*	*Number of Planning Steps*	*Number of Actual Steps*
Perfect Model	Fixed Plan	95	0	95
	Heuristic Search	63	288	63
	A*	68	19,078	68
No Model	Q/Table	116	0	401,593
	Sarsa/CMAC	76	0	94,830
Learned Model	Heuristic Search/CMAC	97	6,067	97
	A*/Linear Model	68	19,204	89
	A*/CMAC	75	6,783,436	1,767

TABLE I. **Comparison of algorithms.** Although all of the methods provide good solutions, modeling the system reduces the number of actual robot moves by allowing trajectory search to be performed offline. Each step is a 0.2 second control time step.

V. Results

Our system uses the parameters given in [1] and also used Euler integration. For the Acrobot system equations, see [11]. We tried additional variations of the CMAC grid size and number of grids for the Sarsa algorithm. The best results were obtained using the parameters described in [1].

Our results are shown in Table I. Given a perfect model, the heuristic methods can determine or search for good trajectories. The techniques that do not attempt to create explicit models must run thousands of iterations on the true robot to find policies. If perfect models are not available, the heuristic search and A*-based methods are able to find good solutions while requiring an order of magnitude less data than Q-Learning approaches. As more domain knowledge used to guide the search, less real data and planning steps are required.

VI. Discussion and Conclusion

We have seen several examples in which domain knowledge and modeling reduce the amount of data required by a learning system. The effectiveness of modeling may be limited if the system is difficult to model, if simulations based on model are prohibitively slow, if imperfect models lead to poor plans, if modeling requires a large amount of data, or if the model structure is unknown. However, for the Acrobot, simple, fast models are available.

Although the idea of a general learning system that requires only minimal domain knowledge is compelling, domain knowledge is present, at least implicitly, at almost every level of learning system design. Parametric models such as polynomials, radial-basis functions, and sigmoidal neural networks make commitments to model structures that are smooth and global. Many general-purpose learning components, such as neural networks, only work well when given good features upon which to train. Features are typically hand-chosen based on domain knowledge. Feature selection is present in all of the algorithms discussed above: the choice of state variables expresses a belief about what the system should learn.

The need for data efficiency argues for domain knowledge to be used throughout the learning system. Indeed, it must be, if the learning agent is to make use of relevant knowledge while learning. This will become increasingly true as learning systems become more intelligent. Like people, advanced learning systems can be expected to take advantage of demonstration and advice from teachers, and to direct their own learning and exploration by asking for relevant information in domain-specific terms. For example, a clever student asks questions within the task domain such as, "Is pumping the only way to swing up to the target height?"

This work can be characterized as demonstrating the utility of learning explicit models to allow mental simulation while learning [2]. Note that because the Q function learns the value of performing actions, Q-learning implicitly builds a model. However, the model cannot be separated from the policy; if the task changes, Q-learning cannot reuse the model to learn a new policy. Thus, another advantage of model building is in task transfer—once a model is learned, it can be reused for planning for new tasks. Comparisons between direct and model-based learning for efficiency and task-transfer can also be found in Atkeson and Santamaria [13] for swing up of pendulum with continuous actions.

Q-learning also implicitly learns the reward function. In the above work, we have assumed that the reward function is known in advance. The planning algorithms knew when they reached the goal, as did the Q-learning agent in [1], even while learning models. An extension to the above work could include

learning the reward function. However, the reward function is the specification of the task and is thus the most necessary kind of domain knowledge.

Acknowledgments

The author thanks Andrew Moore for his suggestion of using grid-based pruning methods for the forward searches.

References

[1] Richard S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding", *in Neural Information Processing Systems 8*, pp. 1038–1044. MIT Press, 1996.

[2] Richard S. Sutton, "DYNA, an Integrated Architecture for Learning, Planning and Reacting", *in Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*, Mar. 1991.

[3] Mark W. Spong, "The swing up control problem for the acrobot", *IEEE Control Systems Magazine*, vol. 15, pp. 49–55, Feb. 1995, Reprinted in Neurocomputing: Foundation of Research.

[4] N. Sadegh and B. Driessen, "Minimum time trajectory learning", *in Proceedings of the American Control Conference*, Seattle, WA, June 1995.

[5] Daniel E. Davison and Scott A. Bortoff, "Regulation of the acrobot", *in Proceedings of the 1995 IFAC Symposium on Nonlinear Control Systems Design*, pp. 390–395, Tahoe City, CA, June 1995.

[6] Michael Athans and Peter Falb, *Optimal Control: An Introduction to the Theory and Its Applications*, McGraw-Hill, New York, 1966.

[7] Christopher J. C. H. Watkins, *Learning from Delayed Rewards*, PhD thesis, University of Cambridge, 1989.

[8] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems", Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[9] Richard Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[10] Stuart J. Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[11] Gary N. Boone, "Minimum-time control of the acrobot", *in IEEE International Conference on Robotics and Automation*, Albuquerque, NM, 1997.

[12] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (cmac)", *Journal of Dynamic Systems, Measurement, and Control*, vol. 23, pp. 220–227, Sep. 1975.

[13] Christopher G. Atkeson and Juan Carlos Santamaria, "A comparison of direct and model-based reinforcement learning", *in IEEE International Conference on Robotics and Automation*, Albuquerque, NM, 1997.