Variable Resolution Reinforcement Learning

Andrew W. Moore

CMU–RI–TR–95–19

A. W. Moore
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

April 1995

Contents

List of Tables

Variable Resolution Reinforcement Learning

Andrew W. Moore
Carnegie Mellon University, School of Computer Science
Pittsburgh, PA 15213
awm@cs.cmu.edu

Abstract

Can reinforcement learning ever become a practical method for real control problems? This paper begins by reviewing three reinforcement learning algorithms to study their shortcomings and to motivate subsequent improvements. By assuming that paths must be continuous, we can substantially reduce the proportion of state space which the learning algorithms need explore. Next, we introduce the *parti-game* algorithm for variable resolution reinforcement learning. In addition to exploring state space, and developing a control policy to achieve a task, parti-game also learns a kd-tree partitioning of state space. Some experiments are described which show parti-game in operation on a non-linear dynamics problems and a path learning/planning task in a 9-dimensional configuration space.

1 Reinforcement Learning

Reinforcement Learning (RL) [Sutton, 1984] is a promising method for robots to program and improve themselves. In this paper we discuss a number of ways to make it more practical. We begin by briefly reviewing three RL algorithms: Q-learning [Watkins, 1989], Dyna [Sutton, 1990] and prioritized sweeping [Moore and Atkeson, 1993, Peng and Williams, 1993], and their performance on a maze problem. We then consider how the addition of some extra assumptions permits further versions of these algorithms which require less exploration of state space and which are more resistant to the curse of dimensionality.

A working RL system would be highly autonomous because it makes only very weak assumptions about the nature of the task to be controlled—that it is a Markov Decision Task. Having made that assumption, the human programmer needs only to define a cost function

$$c : \mathbf{State} \times \mathbf{Action} \rightarrow \Re. \qquad (1)$$

It is then supposedly an entirely automatic process for the RL algorithm to learn the control policy which minimizes the expected sum of future costs

$$E[\sum_{t=\text{now}}^{\infty} c(s(t), a(t))]. \qquad (2)$$

This sum is well-defined if all states can eventually reach an absorbing state of zero cost. There are a number of alternatives to this basic formulation, such as a sum of discounted future costs, a finite horizon sum, and mean future cost. The above equation is conceptually the simplest.

2 Three RL algorithms

In this section we briefly review three reinforcement learning algorithms. We will examine their performance on the maze shown in Figure 1. There are always four actions: North, East, South and West. Each step costs one unit, and the controller does not know in advance where the goal is.

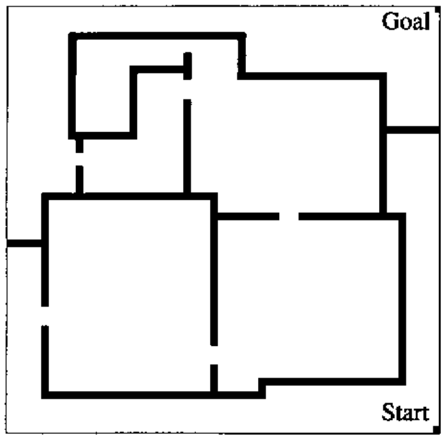

Figure 1: An example maze problem. This uses a 4,096-cell grid, in which some squares are barriers. The start state is the bottom rightmost cell and the goal is the top rightmost.

2.1 Q-learning

In this paper, for ease of exposition, we will restrict the discussion of learning control problems to deterministic systems in which there is a goal state which must be reached with minimum accumulated cost. Q-learning achieves this by maintaining a table of Q-values, indexed by states and actions.

$$Q(s,a) = \begin{array}{l}\text{Estimated minimum cost to goal if}\\ \text{we begin at state } s\text{, apply action } a\text{,}\\ \text{and subsequently perform optimally}\end{array} \tag{3}$$

If the controller is in state s, it chooses the action a to minimize $Q(s,a)$. It then observes the one-step cost $c(s,a)$ of performing the action and the resultant state s'. The estimate of $Q(s,a)$ is now improved to

$$Q(s,a) \leftarrow c(s,a) + \min_{a'} Q(s',a'). \tag{4}$$

The Q table is initialized to zero. Thus, any untried state-action pair will look attractive to the controller. Indeed, during the early stages of learning, Q-learning updates tend to bias the system towards unexplored states. Under repeated trials, this process is guaranteed to converge to optimal [Watkins, 1989, Koenig and Simmons, 1993]. In non-deterministic environments, a more complex update rule is required but convergence to optimal can still be assured.

For our example maze, Q-learning requires 926,682 state transitions before convergence to optimal.

2.2 Dyna

Q-learning takes a long time to converge, even on this simple deterministic example. It can be improved by using extra memory to construct a state-transition graph (a learned "model") of where it has been. One such algorithm is DYNA [Sutton, 1990], which interleaves each physical execution of a state transition with real-time planning. On our problem we permit it to perform 200 steps of asynchronous dynamic programming for every physical step. 200 states are chosen at random, and for each state s, the minimum-cost-to-goal estimate for the state is updated with Bellman's equation:

$$J^*(s) \leftarrow \min_a (c(s,a) + J^*(\texttt{NEXT}(s,a))) \tag{5}$$

where $J^*(s)$ is the estimated minimum-cost-to-goal from s and $\texttt{NEXT}(s,a)$ is the next state in the empirically-determined graph. Dyna takes 39,329 state transitions for the same maze—a large improvement.

2.3 Prioritized Sweeping and Queue-Dyna

Dyna still contains an inefficiency—it applies Bellman updates on *randomly* chosen states. Instead, the states can be selected using a technique called prioritized sweeping [Moore and Atkeson, 1993], which is closely related to an independently invented technique called Queue Dyna [Peng and Williams, 1993]. In these methods, Bellman updates are applied to a fixed number (in our example, 200) of the highest priority states. A state's priority is an estimate of how much it could change if a Bellman update were applied. When a state is updated, and its J^* value subsequently changes by amount Δ, then the immediate predecessors are alerted that they too might change by up to Δ, and are promoted in the priority queue accordingly.

When prioritized sweeping was run on the problem in Figure 1 it required 23,282 steps to achieve an optimal policy.

3 The continuity assumption

It is unlikely that any algorithm could reduce the learning time substantially further. This is because the system is not told in advance what the arc costs are, or where the goal is. Every state-action combination must be tried to ensure it is not a zero-cost arc connected straight to the goal. In our maze example there are approximately 16,000 state-action pairs. For larger control tasks it is entirely impractical to expect the controller to test every state-action pair.

A solution is to tell the controller three things in advance:

- Where the goal is.
- What the one-step cost function $c(s, a)$ is.
- That the only possible paths through state space are continuous.

For a state space discretized into cells, the continuity assumption can easily be incorporated into the same state transition graph that the system builds during the Dyna and prioritized sweeping algorithms. This graph is initialized with the knowledge that the only possible successors of any state are those corresponding to adjacent grid-cells.

This default assumption means that some cells need never be physically explored. Figure 2 shows the set of states which were visited by prioritized sweeping when it was used in conjunction with the continuous paths

Table 1: Number of steps until the optimal path is learned for the maze in Figure 1.

Q-learning	926,682
Dyna	39,329
Prioritized Sweeping	23,282
Prioritized Sweeping + Continuous path assumption	3,772

assumption. The J^* values of all states (visited and unvisited) were computed as before, but all the unvisited states were computed with the default assumption that all neighboring cells could be reached in one step. Even if some neighboring cells are actually inaccessible, this assumption never overestimates J^*. Thus learning consists of beginning with an optimistic assumption that all transitions are attainable, and only ever testing those transitions which could possibly lie on a superior path to the current favorite.

The results on our example problem are summarized in Table 1.

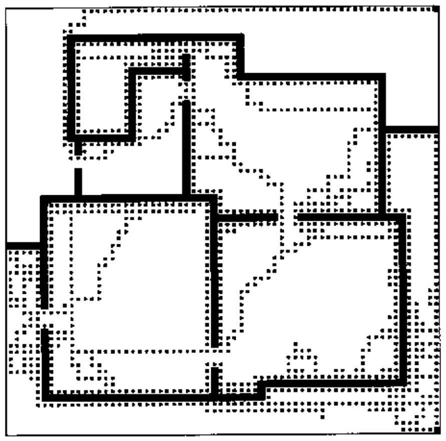

Figure 2: The cells which were visited during learning by prioritized sweeping supplemented with the continuous paths assumption. The start state is the bottom-rightmost cell and the goal is the top rightmost.

4 Variable Resolution Reinforcement Learning

The continuity assumption can reduce the number of states which the robot needs to explore. However, it is still necessary to represent, and reason with, all the states in the grid. Despite the efforts of the previous section, above two or three dimensions this can render reinforcement learning impractical as it falls to the same "curse of dimensionality" that bars Dynamic Programming from high dimensional problems [Bellman, 1957].

We are currently investigating ways of dealing with this problem by learning variable resolution partitionings of state space. Here, we present an algorithm, parti-game, designed for this purpose. The algorithms in the previous section all have non-deterministic counterparts. The version of parti-game reported here is, however, only applicable to deterministic problems. There are two further restrictions:

- A good, but not necessarily optimal, solution must be found in reasonable time.
- A local greedy controller is available, which we can ask to move greedily towards any desired state. There is no guarantee that a request to the greedy controller will succeed. For example, in a maze a greedy path to the goal would soon hit a wall.

4.1 Essentials Of The Parti-Game Algorithm

Here, there is only space to summarize the algorithm. Full detail is given in [Moore and Atkeson, 1994]. The state space is broken into partitions by a *kd*-tree [Friedman *et al.*, 1977]. The controller can always sense its current (continuous valued) state, and can cheaply compute which partition it is in. The space of actions is also discretized so that in a partition with N neighboring partitions, there are N high level actions. Each high level action corresponds to a local greedy controller, aiming for the center of the corresponding neighboring partition.

Each partition keeps records of all the occasions on which the system state has passed through it. Along with each record is a memory of which high level action was used (i.e. which neighbor was aimed for) and what the outcome was. Figure 3 provides an illustration.

Given this database of

$$\text{(partition, high level action, outcome)}$$

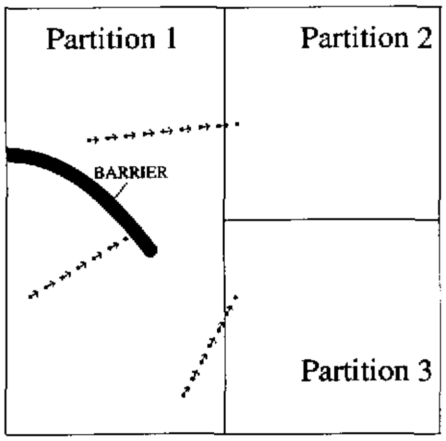

Figure 3: Three trajectories starting in partition 1, using high level action "Aim at partition 2". Partition 1 remembers three outcomes.
`(Part 1, Aim 2` $\rightarrow$ `Part 2)`
`(Part 1, Aim 2` $\rightarrow$ `Part 1)`
`(Part 1, Aim 2` $\rightarrow$ `Part 3)`

triplets, and our knowledge of the partition containing the goal state, we can try to compute the best route to the goal. The standard approach would be to model the system as a Markov Decision Task in which we empirically estimate the partition transition probabilities. However, the probabilistic interpretation of coarse resolution partitions can lead to policies which get stuck. Instead, we use a game-theoretic approach, in which we imagine an adversary. This adversary sees our choice of high level action, and is allowed to select any of the observed previous outcomes of the action. Partitions are scored by minimaxing: the adversary plays to delay or prevent us from getting to the goal and we play to get to the goal as quickly as possible.

As in Section 3, if we have never before attempted an action in a partition we optimistically it will take us to the correct neighbor. This optimism provides an aggressive exploration strategy.

Whenever the system's continuous state passes between partitions, the database of state transitions is updated and, if necessary, the minimax scores of all partitions are updated. If real-time constraints do not permit full re-computation, the updates take place incrementally in a manner similar to prioritized sweeping.

As well as being robust to coarseness, the game-theoretic approach also tells us where we should increase the resolution. Whenever we compute that we are in a losing partition we perform resolution increase, hoping that the route to the goal is hidden by overly coarse resolution. From the continuous paths assumption, we know that any route from our current state to the goal must pass through the border between the sets of losing and non-losing cells. And so it is at the border that we increase the resolution. We first compute the complete set of connected partitions which are also losing partitions. We then find the subset of these partitions which border some non-losing region, and increase the resolution of all these border states by splitting them along their longest axes.

5 Parti-game Experiments

Figure 4 shows a similar maze to our original example in Figure 1. As in the earlier experiments, the algorithm begins with no knowledge of where the obstacles are, and must discover the path to the goal by learning. Figure 5 shows the partitioning of state space, and the resultant path to the goal, after parti-game has finished learning. The final partitioning of state space uses only 344 partitions.

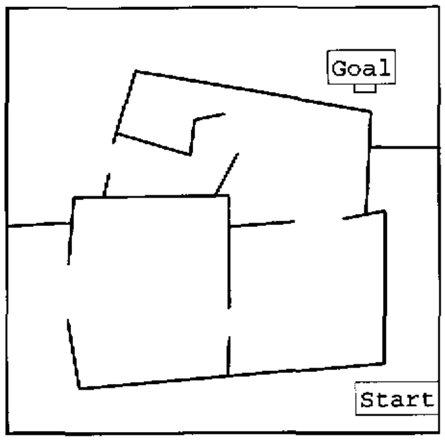

Figure 4: A 2-d continuous maze.

5.1 Non-linear dynamics

Parti-game is not restricted to maze navigation problems. Figure 6 depicts a frictionless puck on a bumpy surface. It can thrust left or right with a maximum thrust of ±4 Newtons. Because of gravity, there is a region near the center of the hill at which the maximum rightward thrust is not strong enough to accelerate up the slope. Thus if the goal is at the top of the slope,

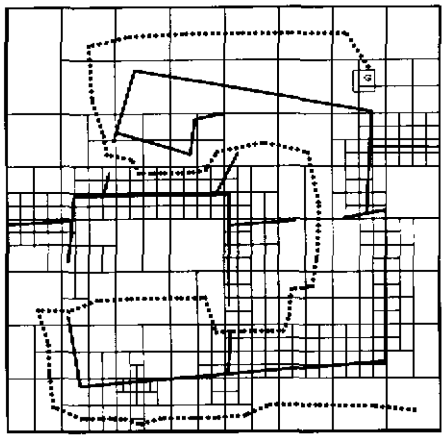

Figure 5: The final partitioning and path for the continuous maze problem.

a strategy which proceeded by greedily choosing actions to thrust towards the goal would get stuck.

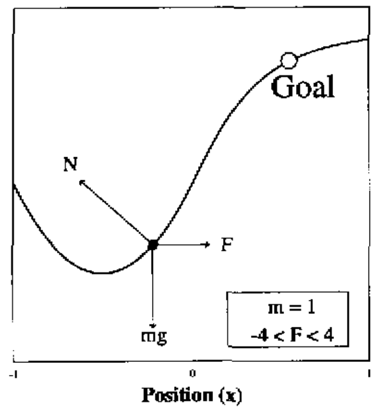

Figure 6: A frictionless puck acted on by gravity and a horizontal thruster. The puck must get to the goal as quickly as possible. There are bounds on the maximum thrust.

This is made clearer in the phase space diagram of Figure 7. The puck's state has two components, the position and velocity. The hairs show the next state of the puck if it were to thrust rightwards with the maximum legal force of 4 Newtons. Notice that at the center of state space, even when this thrust is applied, the puck velocity decreases and it eventually slides leftwards. The optimal solution for the puck task, depicted in Figure 8, is to

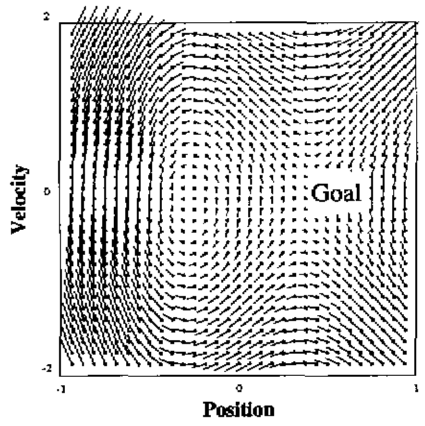

Figure 7: The state transition function for a puck which constantly thrusts right with maximum thrust.

initially thrust away from the goal, gaining negative velocity, until it is on the far left of the diagram. Then it thrusts hard right, to build up sufficient energy to reach the top of the hill.

Figure 9 shows the trajectory through state space during the very first learning trial, while it is exploring and developing its initial partitioning. Figure 10 shows the resulting partitioning and the subsequent trajectory: on its second trial it has already learned the basic strategy of "begin by getting a negative velocity, moving backwards, and only then heading forward with full thrust."

The local greedy controller which parti-game uses is bang-bang. To aim for a partition "north" in state space—a partition with greater velocity—it thrusts with the maximum permissible force of $+4N$. To aim for a lower velocity partition it thrusts with $-4N$. To aim for an "east" or "west" partition, the local controller merely controls its velocity (using a trivial linear controller) to be equal to the velocity of the center of the destination partition. Notice that if the current partition's velocity is greater than zero it is hopeless to greedily aim for the partition on the left. It is also hopeless to aim at the partition on the right if the current partition has negative velocity. In the experiments below, parti-game is given this extra information. Forcing parti-game to learn this from experience approximately doubles the learning time.

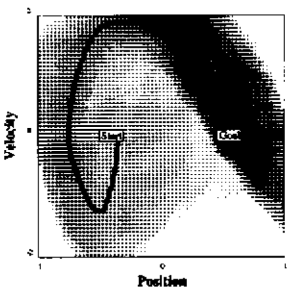

Figure 8: Minimum-time path for the puck on the hill. The optimal value function is shown by dots. The shorter the time to goal, the larger the black dot.

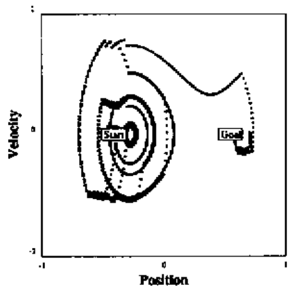

Figure 9: The trajectory of the very first trial, while the system performed its initial exploration of state space.

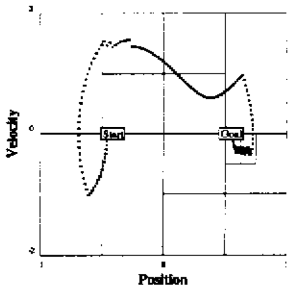

Figure 10: The trajectory and partitioning of the second trial.

5.2 A higher dimensional state space

Figure 11 shows a 9-joint snake-like robot manipulator which must move to a specified configuration on the other side of a barrier. Again, no kinematics model or knowledge of obstacle locations are given: the system must learn these as it explores. It takes seven trials before converging on the solution shown in Figure 12, which requires about two minutes run-time on a SPARC-I workstation. The exploration-length versus final-path-length ratio is 60. Interestingly, the final number of partitions is only 85. This compares very favorably with the 512 partitions which would be needed if the coarsest non-trivial uniform grid were used: $2 \times 2 \times \cdots \times 2$. Unsurprisingly, for the 9-joint snake, this 512 uniform grid is too coarse, and in experiments we performed with such a grid the system became stuck, eventually deciding the problem was impossible.

6 Related work

A few other researchers have attempted to overcome dimensionality problems by decompositions of state space. [Simons *et al.*, 1982] attempted it for 3-degree-of-freedom force control. Their method gradually learned by recording cumulative statistics of performance in partitions. More re-

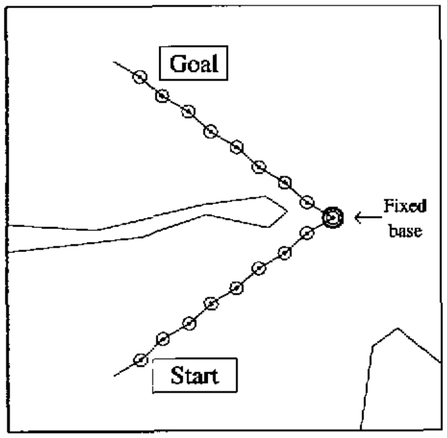

Figure 11: A nine-degree-of-freedom planar robot must move from the shown start configuration to the goal. The joints are shown by small circles on the left-hand diagram which depicts two configurations of the arm: the start position and the goal position. The solution entails curling, rotating and then uncurling. It may not intersect with any of the barriers, the edge of the workspace, or itself.

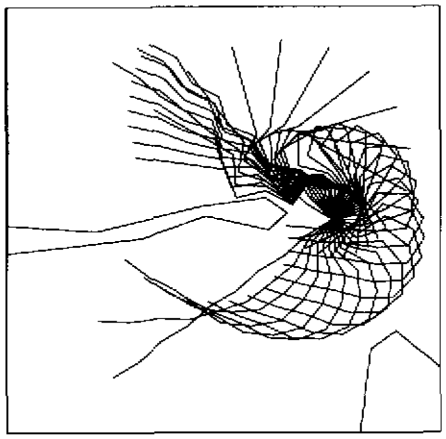

Figure 12: Parti-game's learned solution for the nine-degree-of-freedom snake.

cently, we produced a non-learning method based on variable resolution dynamic programming [Moore, 1991]. [Chapman and Kaelbling, 1991] proposed an interesting algorithm which used more sophisticated statistics to decide which attributes to split. Their objectives were very hard and they obtained only limited empirical success. In [Dayan and Hinton, 1993] a 2-dimensional hierarchical partitioning was used on a grid with 64 discrete squares, and [Kaelbling, 1993] gives another hierarchical algorithm. Both use Q-learning, instead of model-based learning, and also require a predefined decomposition of state space into partitions.

Geometric Decompositions have also been used fairly extensively in Robot Motion Planning (e.g. [Brooks and Lozano-Perez, 1983, Kambhampati and Davis, 1986]), summarized in [Latombe, 1991]. The principal difference is that they assume that a model of the environment is supplied to the system in advance so that there is no learning or exploration capability.

7 Conclusion

This paper began by demonstrating how the removal of some inefficiencies in basic reinforcement learning algorithms can substantially improve the rate of convergence. One of these ideas—assuming paths through state space must be continuous—is the backbone of the splitting criterion of parti-game. There are many interesting avenues which remain open for further investigation.

Acknowledgements

The work was supported by a National Science Foundation Research Initiation Award to Andrew W. Moore.

References

[Bellman, 1957] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[Brooks and Lozano-Perez, 1983] R. A. Brooks and T. Lozano-Perez. A Subdivision Algorithm in Configuration Space for Findpath with rotation. In *Proceedings of the 8th International Conference on Artifical Intelligence*, 1983.

[Chapman and Kaelbling, 1991] D. Chapman and L. P. Kaelbling. Learning from Delayed Reinforcement In a Complex Domain. Technical Report, Teleos Research, 1991.

[Dayan and Hinton, 1993] P. Dayan and G. E. Hinton. Feudal Reinforcement Learning. In S. J. Hanson, J. D Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5.* Morgan Kaufmann, 1993.

[Friedman *et al.*, 1977] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. on Mathematical Software*, 3(3):209–226, September 1977.

[Kaelbling, 1993] L. Kaelbling. Hierarchicial Learning in Stochastic Domains: Preliminary Results. In *Machine Learning: Proceedings of the Tenth International Workshop.* Morgan Kaufmann, June 1993.

[Kambhampati and Davis, 1986] Subbarao Kambhampati and Larry S. Davis. Multiresolution Path Planning for Mobile Robots. *IEEE Journal of Robotics and Automation, Vol. RA-2, No. 3*, 2(3), 1986.

[Koenig and Simmons, 1993] S. Koenig and R.G. Simmons. Complexity Analysis of Reinforcement Learning. In *Proceedings of the Eleventh International Conference on Artificial Intelligence (AAAI-93).* MIT Press, 1993.

[Latombe, 1991] J. Latombe. *Robot Motion Planning.* Kluwer, 1991.

[Moore and Atkeson, 1993] A. W. Moore and C. G. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, 13, 1993.

[Moore and Atkeson, 1994] A. W. Moore and C. G. Atkeson. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multi-dimensional State-spaces. Accepted for Publication in Machine Learning, 1994.

[Moore, 1991] A. W. Moore. Variable Resolution Dynamic Programming: Efficiently Learning Action Maps in Multivariate Real-valued State-spaces. In L. Birnbaum and G. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop.* Morgan Kaufmann, June 1991.

[Peng and Williams, 1993] J. Peng and R. J. Williams. Efficient Learning and Planning Within the Dyna Framework. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior.* MIT Press, 1993.

[Simons *et al.*, 1982] J. Simons, H. Van Brussel, J. De Schutter, and J. Verhaert. A Self-Learning Automaton with Variable Resolution for High Precision Assembly by Industrial Robots. *IEEE Trans. on Automatic Control*, 27(5):1109–1113, October 1982.

[Sutton, 1984] R. S. Sutton. Temporal Credit Assignment in Reinforcement Learning. Phd. thesis, University of Massachusetts, Amherst, 1984.

[Sutton, 1990] R. S. Sutton. Integrated Architecture for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Proceedings of the 7th International Conference on Machine Learning.* Morgan Kaufmann, June 1990.

[Watkins, 1989] C. J. C. H. Watkins. Learning from Delayed Rewards. PhD. Thesis, King's College, University of Cambridge, May 1989.