Reinforcement Learning and Function Approximation[*]

Marina Irodova and **Robert H. Sloan**
mirodo1@uic.edu and sloan@uic.edu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607-7053, USA

Abstract

Relational reinforcement learning combines traditional reinforcement learning with a strong emphasis on a relational (rather than attribute-value) representation. Earlier work used relational reinforcement learning on a learning version of the classic Blocks World planning problem (a version where the learner does *not* know what the result of taking an action will be). "Structural" learning results have been obtained, such as learning in a mixed 3–5 block environment and being able to perform in a 3 or 10 block environment.

Here we instead take a *function approximation* approach to reinforcement learning for this same problem. We obtain similar learning accuracies, with *much* better running times, allowing us to consider much larger problem sizes. For instance, we can train on a mix of 3–7 blocks and then perform well on worlds with 100–800 blocks—using less running time than the relational method required for 3–10 blocks.

Introduction

Traditional Reinforcement Learning (RL) is learning from interaction with an environment, in particular, learning from the consequences of actions chosen by the learner (see, e.g., (Mitchell 1997; Kaelbling, Littman, & Moore 1996)). The goal is to choose the best action for the current state. More precisely, the task of RL is to use observed rewards to learn an optimal (or almost optimal) policy for the environment. RL tasks are challenging because often the learner will execute a long sequence of actions and receive reward 0 for every action but the last one in the sequence.

(Dzeroski, Raedt, & Driessens 2001) introduced Relational Reinforcement Learning (RRL). RRL seeks to improve on RL for problems with a relational structure by generalizing RL to relationally represented states and actions (Tadepalli, Givan, & Driessens 2004b). Traditionally, RL has used a straightforward attribute-value representation, especially for the common Q-learning strategy. For instance, Dzeroski et al. showed how, using RRL, one could learn on a group of different small-size problem instances, and perform well on more complex, or simply larger problem instances having similar structure. Using a traditional attribute-value representation for RL this would be impossible.

Dzeroski et al. and later (Gärtner, Driessens, & Ramon 2003) used a learning version of the classic Blocks World planning problem to test their ideas using a Q-learning approach to RL. In this learning version, introduced originally by (Langley 1996), the task is finding the optimal policy in the blocks world, where, unlike the planning version, *the effects of the actions are not known to the agent* (until after the agent takes the action and observes its new state). We use this same test problem in our work here. We will be comparing our learning rates especially to Gärtner et al. who improved on the learning rates of Dzeroski et al. by using a clever new approach for RRL: graph kernels (Gärtner, Flach, & Wrobel 2003) and Gaussian Learning.

Q-learning is a common model-free strategy for RL; it was used for RRL by both Dzeroski et al. and Gärtner et al. and we also use it. Q-learning maps every state-action pair to a real number, the Q-value, which tells how optimal that action is in that state. For small domains this mapping can be represented explicitly (Mitchell 1997); for large domains, such as a 10-block blocks world (which has tens of millions of states), it is infeasible. Even if memory were not a constraint, it would be infeasible to train on all those state-action pairs. RRL is one approach to this problem.

Here, instead, we take a *function approximation* approach by representing each state as a small fixed number of features that is independent of the number of blocks. Our main goal is to get similar learning accuracy results with (much) better running time than RRL. This allows us to perform in extremely large state spaces. Thus we can demonstrate tremendous generalization: training on 3–7 blocks instances and performing on up to 800 blocks. The earlier works never considered more than 10 blocks, and really could not, given their running time.

There has been relatively little work on using function approximation for model-free RL in general, and Q-learning in particular. The idea is sketched in (Russel & Norvig 2003) and is used by (Stone & Veloso 1999) (in a setting with other complicating factors). (Of course the use of function approximation in machine learning in general is not novel at all.)

The paper is organized as follows. First, we talk about about the problem specification, briefly describing RL and

[*]This material is based upon work supported by the National Science Foundation under Grant Nos. CCR-0100036 and CCF-0431059.

Q-learning. Then we introduce Blocks World problem in RL and explain the representation of the symbolic features for this domain. In the next section, we present our experiments and results. Finally, we devote a short section to additional related work, and then we conclude.

Problem Specification

Following (Dzeroski, Raedt, & Driessens 2001; Gärtner, Driessens, & Ramon 2003), we consider the following form of RL task for the learning agent:

What is given:

- Set S of possible states.
- Set A of possible actions.
- ***Unknown*** transition function $\delta : S \times A \rightarrow S$.
- Reward function r, which is 1 in *goal* states and 0 elsewhere. In the problem considered here, the goal states are also terminal.

What to find:

- An optimal policy $\pi^* : S \rightarrow A$, for selecting next action a_t based on the current observed state s_t.

A common approach is to choose the policy that produces the greatest cumulative reward over time. The usual way to define it is $V^\pi(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ where $\gamma \in [0, 1)$ is a *discount factor*. In general, the sequence of rewards r_{t+i} is obtained by starting at state s_t, and by iteratively using the policy π to choose actions using the rule $a_t = \pi(s_t)$. For the goal-state type of reward, in one *episode* a single reward of 1 is obtained at the end of the episode, and the objective is to get to that goal state using as short an action sequence as possible. Formally, we require the agent to learn a policy π that maximizes cumulative reward V^π. This policy will be called *optimal* and denoted by π^*.

Q-learning

Q-learning is a very common form of RL. In Q-learning, the agent learns an action-value function, or *Q-function*, given the value of taking a given action in a given state. Q-learning always selects the action that maximizes the sum of the immediate reward and the value of the immediate successor state.

The Q-function for the policy π:

$$Q(s, a) = r(s, a) + \gamma V^\pi(\delta(s, a)) \ .$$

Therefore, the optimal policy π^* in terms of Q-function:

$$\pi^*(s) = \arg\max_a [Q(s, a)]$$

The *update rule* for the Q-function is defined as follows:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)),$$

which is calculated when action a is executed in state s leading to the state s'. Here $0 \leq \alpha \leq 1$ is a *learning rate* parameter and $gamma$ is the discount factor. (See, e.g., (Mitchell 1997), (Russel & Norvig 2003) for more details)

Function Approximation and Feature-Based Method

It may be very difficult in general to learn a Q-function perfectly. We often expect learning algorithms to get only some approximation to the target function. We will use *function approximation*: we will learn a representation of the Q-function as a linear combination of features, where the features describe a state. In other words, we translate state s into the set of features $f_1, \ldots, f_n$, where n is the number of features.

A somewhat novel approach is that we consider a small set of symbolic, feature-based, actions, and we learn a distinct Q-function for each of these actions. Thus we have the set of Q-functions:

$$Q^a(s, a) = \theta_1^a f_1 + \cdots + \theta_n^a f_n \tag{1}$$

for each symbolic action a. The update rule is

$$\theta_k^a = \theta_k^a + \alpha[r + \gamma \max_{a'} Q^a(s', a') - Q^a(s, a)] \frac{dQ^a(s, a)}{d\theta_k^a} \tag{2}$$

In Algorithm 1, we sketch the Q-learning algorithm we used; it generally follows (Mitchell 1997).

Algorithm 1 The RL Algorithm

```
initialize all thetas to 0
repeat
    Generate a starting state $s_0$
    i=0
    repeat
        Choose an action $a_i$, using the policy obtained from
        the current values of thetas
        Execute action $a_i$, observe $r_i$ and $s_{i+1}$
        i=i+1
    until $s_i$ is terminal (i.e., a goal state)
    for $j = i - 1$ to $0$ do
        update the value of thetas for all taken actions using
        (2)
    end for
until no more episodes
```

Commonly in Q-learning, the training actions are selected probabilistically. In state s our probability of choosing action a_i is

$$Pr(a_i|s) = \frac{T^{Q^i(s,a_i)}}{\sum_{\text{all legal (in } s\text{) actions } a} T^{Q^a(s,a_k)}} \tag{3}$$

The parameter T, called "temperature," makes the exploration versus exploitation tradeoff. Larger values of T will give higher probabilities to these actions currently believed to be good and will favor exploitation of what the agent has learned. On the other hand, lower values of T will assign higher probabilities to the actions with small Q values, forcing the agent to explore actions with low probability (see (Mitchell 1997) for more details).

Blocks World as RL Problem

We now describe Langley's RL version of the classic Blocks World Problem (Langley 1996) that we mentioned in the introduction.

A Blocks World contains a fixed set of blocks. Each block can have only two positions: on the *top* of another block or on the *"floor"*. The available action in this world is *move(a,b)*, where a is a block and b is either a block or the "floor." In the earlier related work and also in our work here, three goals are considered:

Stack States with all blocks in a single stack

Unstack States with every block on the floor

On(a, b) States with the block with label a on top of the block labeled b

The On(a, b) goal is different from the other two goals because it requires labels on the blocks, and therefore we, like the earlier authors, will treat the On(a, b) goal a little differently from the other two goals.

For the Stack and Unstack goals we use 7 features, independent of the number of blocks. They are the number of: blocks, stacks, blocks in the highest stack, highest stacks, blocks in the second highest stack, second highest stacks, stacks of height one.

For the On(a, b) goal we use additional information about the positions of blocks a and b, again independent of the number of blocks.

In a full, attribute-value representation of the state space, there would in be n^2 actions for an n-block instance. These actions would be of the form move(x, y), where x was any of the n blocks and $y \neq x$ was either the floor or any of the $n - 1$ remaining blocks. Of course, in any given state only some of those n^2 moves would be possible.

We used a much smaller fixed set of actions independent of the number of blocks, which are connected to our set of features. Now in Table 1 we specify this set of legal, symbolic, feature-based actions for the Stack and Unstack goals. There are 11 of them:

	One	Middle	Tallest	Floor
One	✓	✓	✓	
Middle	✓	✓	✓	✓
Tallest	✓	✓	✓	✓

Table 1: All possible actions for Stacking and Unstacking.

That is, the 11 legal actions are *OneToOne*, *Tallest-ToFloor*, etc. By *OneToOne* we mean move any block in a stack of height one on top of any block of height one. The choice to leave out the "no-op" action of moving a block in a one-block stack to the floor was arbitrary; learning results and running times are similar with it allowed or not.

Notice that we learn 11 different Q-functions. For example, for the action *OneToOne*, if we denote that action by 1, then the corresponding Q-function is

$$Q^1(s, a_1) = \theta_1^1 f_1 + \cdots + \theta_n^1 f_n,$$

where f_i for $i = 1 \ldots n$ are the features mentioned above. When we train and test, and select one of those 11 actions, we then make an arbitrary translation of it to some particular move(x, y) action.

Each action has its own set of thetas. In other words, for stacking and unstacking there are 77 thetas that have to be learned. For the On(a, b) goal, Table 1 would have two additional rows and columns for a and b, and there would be correspondingly more thetas.

Note that the number of states increases exponentially with the number of blocks. Our technique allows us to avoid explicitly considering these individual states.

Experiments and Results

Of the two earlier works, (Gärtner, Driessens, & Ramon 2003) obtained better learning accuracy. They trained on a mixture of 3, 4, and 5 block instances, together with some 10-block guided traces (i.e., solutions to 10-block worlds provided by a human or a traditional planner, not by Q-learning in a 10-block world. Note that the 10-block world has so many states that Q-learning's strategy of random exploration is very unlikely ever to find a goal state during training.). They then tested the learned Q function on randomly generated starting states in worlds with 3 to 10 blocks. We performed the same experiment for our method, except that we did not need the guided traces.

We next describe how we set parameters; then we give the experimental results.

Parameter settings

As with the two RRL papers we are comparing to, and indeed, as with all Q-learning, there are several parameters that must be set. We have six parameters (a typical number), three (discount rate, learning rate, and temperature) that are common to almost all Q-learning, and three (another exploitation versus exploration parameter and two parameters for weighting features) that are not. We do not have some of the parameters that the earlier RRL papers used (e.g., graph kernel parameters).

We used the parameter-setting method of (Gärtner, Driessens, & Ramon 2003), basically making test runs to fix the parameters to seemingly good values, and then starting learning over with those settings and new random initializations.

Seemingly all Q-learning uses a *discount factor* $\gamma = 0.9$; we also did.

The *learning rate*, α, must be in the range 0 to 1. Results were somewhat sensitive to this parameter; for good results we needed to use a different value for On(a, b) than for the other two goals.

Exploitation versus exploration parameters There are two of these.

- For the *temperature* T we used an initial value of 10 and incremented by 1 after each episode. Our results were very similar for any initial temperature up to 300.
- *The number of times when an action is chosen randomly in one run.* At the very start of learning, when the Q-values are initialized to an arbitrary value, it is easy to get stuck in a local minimum. Thus initially, we choose the

actions randomly until the agent had tried every action a certain number of times. We set this number to 10.

Feature normalization In a function-approximation, feature-based approach, the features must be normalized.

- The features are used to calculate the Q-values, which in turn are used as an exponent in Equation (3), so they need to be kept not to large. We divided by a normalization factor that depended on the number of blocks, number of features and maximum sum over all sets of features.
- *Weight factor for the boolean features*. This parameter is used only for the goal On(a, b). For this goal, some features are Boolean (i.e., always either 0 or 1), whereas the rest can take values up to the number of blocks (and we considered performance on up to 800 blocks). For example, some Boolean features are: "Is block a (respectively b) on the floor?" and "Is the top of block a (respectively b) clear?"

  These Boolean features carry an important information, but when the Q-value of the state-action pair is calculated, the Boolean features do not have much influence, especially when the number of blocks is large. Therefore we decided to weight them by multiplying each Boolean feature by the number of blocks.

Experiment: generalizing across changing numbers of blocks

We use a set of experiments introduced in (Dzeroski, Raedt, & Driessens 2001), and the only set of experiments reported by (Gärtner, Driessens, & Ramon 2003). There are separate runs and measurements for each of the three goals Stack, Unstack, and On(a, b). These experiments were a significant achievement of RRL, because the training instances had different numbers of blocks, and the test instances had different numbers of blocks, including some numbers not seen at all in the training set. A naive attribute-value approach to RL simply could not do this.

Following the earlier work, for these experiments run started with 15 episodes with three blocks, followed by 25 episodes with four blocks, and then 60 episodes with five blocks. Then the learned policy was evaluated for a mixed number of blocks as was described earlier. Each run consisted of 100 episodes. The results were averaged over five runs. The results are shown in Figure 1.

We used the same measure of learning accuracy as Gärtner et al., which was also one of the measures used by Dzeroski et al. That is, we take a random sample of states. The quality measure for this set of experiments is the percentage of the states in the sample for which an optimal plan is generated, called "average reward" for short. It ranges from 0 to 1.

Analysis: After about 60 episodes, we had essentially perfect performance for the stack and unstack goals, and performance of roughly 97% for the more difficult On(a, b) goal.

We note that the results were sensitive to the value of the learning rate parameter α. We pick α using methodology of (Gärtner, Driessens, & Ramon 2003): we ran our experiments several times with different settings of parameters and chose the best ones.

Comparison with (Gärtner, Driessens, & Ramon 2003) and (Dzeroski, Raedt, & Driessens 2001) We used similar tests as (Dzeroski, Raedt, & Driessens 2001) and (Gärtner, Driessens, & Ramon 2003) papers, including their method of choosing parameters. The graph kernel method has the better learning accuracy of these two previous RRL methods (Gärtner, Driessens, & Ramon 2003) for this problem. It is essentially perfect for stack after 250 episodes and for unstack after 400 episodes. For On(a, b), average reward of 90% is obtained after 1,000 episodes.

Our method is obtaining essentially perfect performance for stack and unstack after 60 episodes, and 97% average reward for On(a, b). As we shall see, our time computer running time per episode is *drastically* less.

Furthermore, we were actually doing more generalization, in the following sense. We relied exclusively on training on worlds with 3, 4, and 5 blocks. (Gärtner, Driessens, & Ramon 2003) also used some "guided traces" for 10-block instances in their training.

First, however, we describe some experiments that our much faster running time allows us to run.

Experiment: Large number of blocks

The two previous papers on RRL in Blocks World never considered more that 10 blocks, because of the running time, which increases dramatically with the number of blocks in the state.

The purpose of these tests is to show how the learned policy works in the really large domains.

Test 1: Now we train on a mix of three, four, and five block instances, and test on instances with 10–80 blocks (instead of 3–10 blocks).

Again, we had one experiment for each of our three goals. We first tried runs of 100 episodes for each of the three goals. Each run started with 15 episodes with three blocks, followed by 25 episodes with four blocks, and then 60 episodes with five blocks. The learned policy was then evaluated for a mixed number of blocks. We randomly generated a sample of 156 states (the number used in the previous experiments) having 10 to 80 blocks.

Analysis: For the stack and unstack goals the performance was optimal: the number of steps to the goal states obtained by using our policy equals to the optimal number of steps.

However, for the goal On(a, b) this performance is $\approx 50\%$. After increasing the number of episodes up to 600 and adding 6 block instances to the learning, we got better performance: $\approx 80\%$.

Test 2 (really big): We evaluated the learned policy on a much larger number of blocks: 100–800. For the stack and unstack goals we trained on 100 episodes. We obtained optimal performance for stack and good performance (better than 95%) for the other two goals, although it was really necessary to use more blocks and episodes in the training. To get a good result for the On(a, b) goal we had to increase the number of episodes up 1500 and the number of blocks we were learning on to seven. In this case, our experiment ran as follows: Each run started with 15 episodes

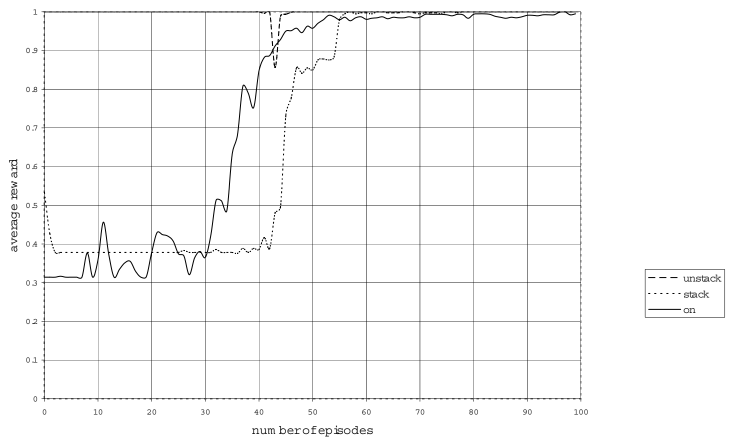

Figure 1: Learning curves for all 3 goals. Average reward is percentage of optimal plans generated for a sample of 156 starting states with 3–10 blocks.

with three blocks, followed by 25 episodes with four blocks, 160 episodes with five blocks, 200 with six, and the rest with seven. Learning curves are shown in Figure 2 and running times in Table 3.

Efficiency

In this subsection we discuss the running time of our learning algorithm, and compare it to the running time of (Dzeroski, Raedt, & Driessens 2001), which is the faster of the two previous RRL algorithms (Driessens 2004). As we stated above, the number of the states increase dramatically with a number of blocks. There is table in (Dzeroski, Raedt, & Driessens 2001) paper, which shows the number of states and number of reachable goal states for three goals and different number of blocks. For example, according to this table, if the number of blocks is 7, then the number of states is 1546, but for 10 blocks this number is 1441729.

To compare to Dzeroski et al., we consider training time on a fixed number of blocks (an experiment that Gärtner et al. did not run) in Table 2 and on a variable number of blocks in Table 3.

Goal	3	4	5	6
Stacking	1.031	0.972	1.192	1.492
Unstacking	0.671	0.801	1.202	1.482
On(a, b)	1.291	1.312	1.742	1.873

Table 2: Training time in seconds on a Dell laptop with a 2.40 GHz CPU on fixed number of blocks. The time was recorded after one run of 100 episodes.

This table shows, that the running time is much better than (Dzeroski, Raedt, & Driessens 2001) paper. For 30 episodes on 3 blocks they required 6.25–20 minutes, whereas we used roughly 1 second for 100 episodes. (However, take into account that they used Sun Ultra 5/270 machines versus our using a Dell Latitude 100L with a 2.40 GHz Celeron CPU, so the difference in hardware probably accounted for a 8–10-fold speedup.)

The second table displays the running time on the experiments whose results were reported by Gärtner and that we discussed here—training time for mixed numbers of blocks. The left column is the time for 100 episodes on a mixture of 3–5 blocks and is directly comparable to the earlier work. The right column is the training time for 1,500 episodes on a mixture of 3–7 blocks (for performance on 100–800 blocks). For the smaller case we needed roughly 2 seconds for 100 episodes. For comparison, to run 45 episodes learning on varying number of blocks (from 3 to 5) (Dzeroski, Raedt, & Driessens 2001) spent 4–12 *hours*, depending on the goal.

Goal	3-5 (100)	3-7 (1500)
Stacking	1.7	17.0
Unstacking	1.7	19.7
On(a, b)	2.3	26.1

Table 3: Training time on mixed number of blocks after 1 run with 100 and 1500 episodes (in seconds on a Dell laptop with a 2.40 GHz processor).

Our running time seems to scale linearly or even better than that with number of episodes; running times for 1500 episodes were only in the range 17–26 seconds.

Additional related work

There are two main approaches to RL: model-free (including Q-learning) and model-based. There has been some work on handling very large state spaces with model-based approaches (Sallans & Hinton 2004; Tsitsiklis & Roy 1996). A variation of Q-learning called HAMQ-learning that learns without performing the transition function is discussed in (Parr & Russell 1997).

A recent overview of RRL in general is (Tadepalli, Givan, & Driessens 2004a).

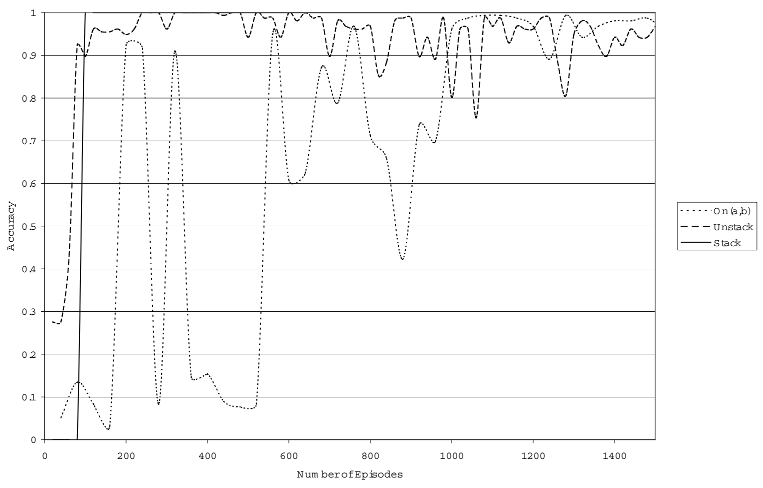

Figure 2: Learning curves for all 3 goals with 100–800 blocks.

The Blocks World problem is famous in planning. There is a web-site devoted to the blocks world problem at `http://rsise.anu.edu.au/~jks/bw.html`.

Conclusions and future work

In this paper we proposed a new method for using RL with Q-learning approach for relational problems that effectively reduces the state space by encoding each state in a compact feature. The conversion from the original state to the set of features can be done extremely fast. Once in this representation, learning is very fast. Our approach maps every state-action pair to Q-value. This method has been chosen for its ability to approximate Q-functions giving optimal or almost optimal performance with extremely good running time.

Experiments in the Blocks World show somewhat better performance than previous work in dramatically less time. Also the learned policy can be applied to a domain with a much larger number of blocks (up to 800) and the time spent on getting an almost optimal policy is within seconds.

We plan to extend our approach to more complex domain: the Tetris game, which has been considered by others working on RL (see, e.g., (Tsitsiklis & Roy 1996)). Some of the features we will use are: height of the wall (or highest stack), number of holes, deepest canyon. The set of actions (low level) will include: rotate, move, drop. game.

References

Driessens, K. 2004. Peronsal communication.

Dzeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43(1/2):5–52.

Gärtner, T.; Driessens, K.; and Ramon, J. 2003. Graph kernels and Gaussian processes for relational reinforcement learning. In *ILP03*, volume 2835 of *LNAI*, 146–163. SV.

Gärtner, T.; Flach, P.; and Wrobel, W. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel*, volume 2777 of *Lecture Notes in Artificial Intelligence*, 129–143. Springer.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. P. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–385.

Langley, P. 1996. *Elements of Machine Learning*. San Matco, CA: Morgan Kaufmann.

Mitchell, T. 1997. *Machine Learning*. Boston: McGraw-Hill.

Parr, R., and Russell, S. 1997. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10. MIT Press.

Russel, S., and Norvig, P. 2003. *Artificial Intelligence a modern approach*. Upper Saddle River, NJ: Prentice Hall, 2nd edition.

Sallans, B., and Hinton, G. E. 2004. Reinforcement learning with factored states and actions. *J. Mach. Learn. Res.* 5:1063–1088.

Stone, P., and Veloso, M. M. 1999. Team-partitioned, opaque-transition reinforcment learning. In *RoboCup-98: Robot Soccer World Cup II*, 261–272.

Tadepalli, P.; Givan, R.; and Driessens, K., eds. 2004a. *Proc. Workshop on Relational Reinforcement Learning at ICML '04*. Available on-line from `http://eecs.oregonstate.edu/research/rrl/`.

Tadepalli, P.; Givan, R.; and Driessens, K. 2004b. Relational reinforcement learning: An overview. In Proc. Workshop on Relational Reinforcement Learning at ICML '04, available on-line from `http://eecs.oregonstate.edu/research/rrl/`.

Tsitsiklis, J. N., and Roy, B. V. 1996. Feature-based methods for large scale dynamic programming. *Machine Learning* 22:59–94.