Incorporating Prior Knowledge and Previously Learned Information into Reinforcement Learning Agents

Kevin R. Dixon Richard J. Malak Pradeep K. Khosla

Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{krd,rmalak,pkk}@cs.cmu.edu

Prepared for the *Institute for Complex Engineered Systems Technical Report Series*, 2000.

31 January, 2000

Abstract

Reinforcement learning has received much attention in the past decade. The primary thrust of this research has focused on *tabula rasa* learning methods. That is, the learning agent is initially unaware of its environment and must learn or re-learn everything. We feel that this is neither realistic nor effective. While the agent may start out with little or no knowledge of its environment, it must be able to incorporate new information into the learning of subsequent tasks otherwise the learning effort is largely wasted. To address the shortcomings of *tabula rasa* learning, we present a general and intuitive approach for incorporating previously learned information and prior knowledge into the reinforcement learning process. We demonstrate the potential of this method on learning problems in the mobile-robot and grid-world domains, where results indicate that learning time can be decreased. We also demonstrate that multiple knowledge sources can be incorporated into the learning process.

1 Introduction

Incorporating prior knowledge and previously learned information in machine learning tasks is a subject that is receiving increased attention. In these problems, a learning agent attempts to design a controller that maximizes some performance metric. For many tasks, *tabula rasa* learning may not be appropriate. The designer of a system may have some *a priori*, domain-specific knowledge or the agent itself may have already learned a task that may prove useful to the problem at hand. A method for incorporating this knowledge into a reinforcement learning controller, which learns by actively exploring its environment, would be particularly useful.

Through the past forty years, the main thrust of machine learning research has been toward improving the performance of *tabula rasa* systems. Many sophisticated and powerful techniques have been developed that allow a machine to learn a task quickly. There are several noteworthy examples of machines surpassing the performance of their human creator (e.g., (Tesauro & Sejnowski, 1992) and (Samuel, 1959)). However, these programs usually learn the task *tabula rasa*. This is not always desirable; the designer of a system may have some *a priori*, domain-specific knowledge or the agent itself may have already learned a task that may prove useful to the problem at hand. Oftentimes, *tabula rasa* learning is utilized because the internal representations in the learning algorithm are not well understood. Thus, incorporating prior knowledge into such systems can be extremely difficult (Baxter, 1994). Recently, researchers have started to focus on learning agents exploiting previously learned information (Pratt, 1994).

We propose a method for embedding previously learned information and prior knowledge into the controller of a reinforcement learning agent. This method utilizes an intuitive representation, scales well to large problems and allows for easy analysis.

In this paper, we give an introduction to reinforcement learning in Section 2, describe our method for embedding prior knowledge in Section 3, apply our method to a couple problems and give experimental results in Section 4, discuss related work in Section 5, and state our conclusions and future work in Section 6.

2 Reinforcement Learning

In the reinforcement learning (RL) paradigm, an agent exists in an environment, whether embodied in the real world or in a simulated world (Kaelbling et al., 1996). The agent can sense the state of the environment through sensors and affect its environment by executing actions through actuators. Furthermore, at every discrete time step, the agent receives an external scalar reward signal from the environment (see Figure 1).

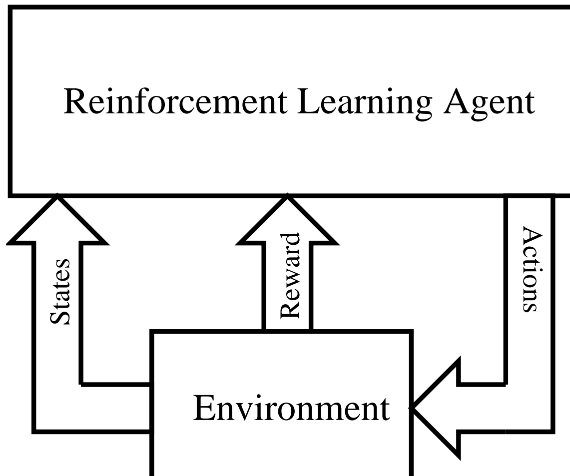

Figure 1: Conceptual diagram of a RL agent.

2.1 Markov Decision Processes

In reinforcement learning algorithms, the environment is implicitly assumed to have the Markov property. The state-to-state transition probabilities of the underlying Markov chain are conditioned on the action executed. This type of system is called a Markov Decision Process (MDP). A MDP is specified by a finite set of states, S, a finite set of actions, A, and the state-to-state transition probabilities, $p : S \times A \times S \mapsto \Re_{[0,1]}$. In words, $p(1, 0, 2)$ is the probability of entering state 2 by executing action 0 in state 1. MDPs also have a scalar reward function, $r : S \times A \times S \mapsto \Re$. In words, $r(2, 1, 0)$ is the reward for entering state 0 by executing action 1 in state 2. A visualization of a simple MDP is given in Figure 2. At each discrete time step, the agent executes an action given by a policy, $\pi : S \mapsto A$. The *model* of the environment is considered to be the functions p and r, since these functions describe how the actions of an agent affect the environment and the reward that the agent receives.

Markov Decision Processes with reward functions give rise to two natural value functions. First is the state value function, $V : S \mapsto \Re$, defined as the expected sum of discounted future rewards for a given state and policy

$$V^{\pi}(s_0) \triangleq \sum_{t=0}^{\infty} E_{s_{t+1}} \left\{ \gamma^t r(s_t, \pi(s_t), s_{t+1}) \middle| \pi, s_0 \right\}$$

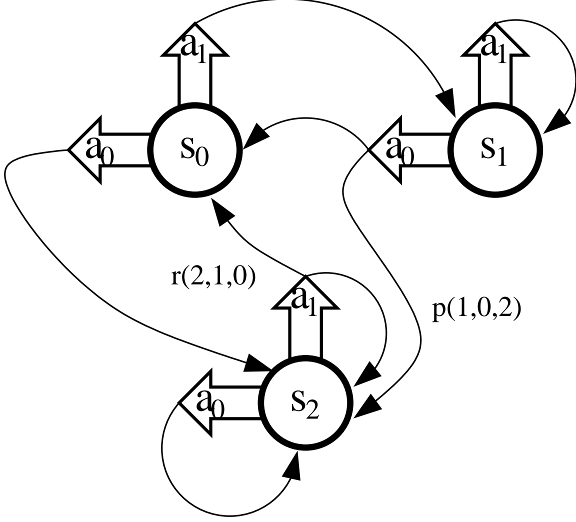

Figure 2: A three-state MDP with two actions in each state.

$$= \sum_{t=0}^{\infty} \sum_{s_{t+1} \in S} \gamma^t p(s_t, \pi(s_t), s_{t+1}) r(s_t, \pi(s_t), s_{t+1})$$

Where $\gamma \in (0, 1]$ is a discount factor, similar to an inflation rate. To the agent, future rewards are not as great as immediate rewards when $\gamma < 1$. Intuitively, $V^{\pi}(s_0)$ is the amount of reward the agent expects to receive starting from state s_0 until the end of time, under the policy π.

The other value function is called the Q-function. This function, $Q : S \times A \mapsto \Re$, is defined similarly to the state value function

$$Q^{\pi}(s_0, a) \triangleq E_{s_1} \left\{ r(s_0, a, s_1) + \gamma V^{\pi}(s_1) \middle| \pi, s_0, a \right\}$$
$$= \sum_{s_1 \in S} p(s_0, a, s_1) \Big(r(s_0, a, s_1) + \gamma V^{\pi}(s_1) \Big)$$

Intuitively, $Q^{\pi}(s_0, a)$ is the amount of reward the agent expects to receive by executing action a starting in state s_0 and then following the policy π until the end of time.

The optimal value functions are denoted by

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in S$$
$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a), \forall s \in S$$

For non-pathological MDPs, there exists an optimal policy

$$\pi^*(s) = \arg\max_{a} Q^*(s, a), \forall s \in S$$

that may not be necessarily unique (Bertsekas & Tsitsiklis, 1996).

2.2 Computing the Optimal Policy

Several techniques exist for computing the optimal policy, π^*. The two most popular methods are Dynamic Programming (DP) (Bellman, 1957) and reinforcement learning. DP requires an exact model of the environment (the functions p and r) and finds the globally optimal solution and RL does not require a model of the environment and finds a locally optimal solution. RL tends to scale better to larger real-world problems than DP for these reasons. Thus for this work, we will consider only RL algorithms to compute policies.

As stated earlier, the environment in reinforcement learning algorithms is implicitly assume to be a Markov Decision Process and a model of the environment is not required. Several algorithms exist that can compute an optimal policy. Q-Learning (Watkins, 1989) has received the bulk of attention by researchers, most likely for its simplicity and convergence proofs, though the guarantees on convergence are not practical. Q-Learning iteratively approximates the Q-function as follows

$$\hat{Q}(s, a) \leftarrow (1-\alpha)\hat{Q}(s, a) + \alpha \Big(\rho + \gamma \max_{\tilde{a}} \hat{Q}(\tilde{s}, \tilde{a}) \Big) \quad (1)$$

Where $\alpha \in (0, 1]$ is a step-size parameter, γ is the discount rate, ρ is the reward received from the environment (this should be an unbiased sampling of the reward function r), and $\tilde{s}$ is the resultant state after the current time step. Watkins showed that $\hat{Q}(s, a)$ from (1) is an asymptotically unbiased estimator of $Q^*(s, a)$ for realistic assumptions. Thus, the following policy will be optimal in the limit

$$\hat{\pi}(s) = \arg\max_{a} \hat{Q}(s, a)$$

Note that a model of the environment (the functions p and r) is not used to compute this unbiased estimate. This means that with no prior knowledge or dynamical model, an agent can compute the optimal policy. In practice, however, we usually settle for a locally optimal policy due to time considerations. Since no model of the environment is used in RL, the environment must be explored through trial-and-error so that the samples of the reward signal, ρ, approximate the true expectation of the reward function, r.

2.3 Reinforcement Learning Controllers

In reinforcement learning, the controller of an agent is either on-policy or off-policy. On-policy controllers (Figure 3) execute actions that affect the environment. Thus, the RL algorithm is determining which actions

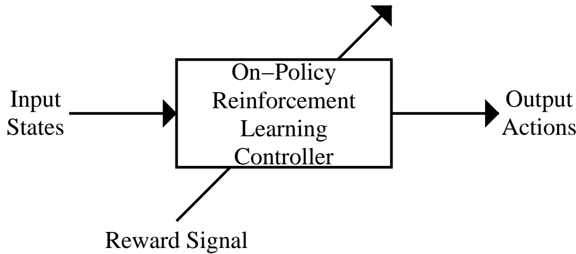

Figure 3: An on-policy RL controller executes actions that affect the environment.

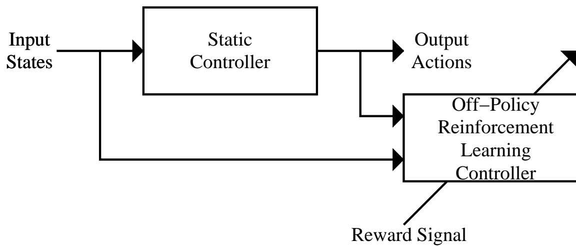

Figure 4: An off-policy RL controller can compute an optimal policy but the static controller executes actions that affect the environment.

the agent will execute in a given state. Off-policy controllers (Figure 4) observe the actions that a static controller executes and do not determine which actions the agent will execute. The off-policy controller is not attempting to mimic the static controller, but determining which actions will maximize the reward signal from the state-action examples the static controller presents it.

2.4 Generalization and Function Approximators

Typically, generalization across similar states is desired. To this end, controllers normally represent the value function in a parameterized function approximator, such as an artificial neural network (ANN). This may allow the agent to visit the states less often to get a good estimate of the expectation of the reward function, r, from the samples, ρ. However, there are no convergence results for either on- or off-policy controllers when non-linear approximators (e.g., sigmoidal multilayer perceptrons) are used. Furthermore, the use of function approximation, even linear mappings, may result in divergence or oscillation in off-policy controllers (Bradtke, 1993; Baird, 1995; Boyan & Moore, 1995; Tsitsiklis & Van Roy, 1995). The fundamental cause of divergence is that the states the static controller visits are drawn from a different distribution than the off-policy controller would generate. Thus, as the distributions become further apart, the off-policy controller may become more likely to diverge. However, researchers have continued to use function approximators since the benefits of generalization normally out-weight these potential problems.

2.5 Exploration of the Environment

Since a reinforcement learning agent must explore its environment through trial-and-error, many methods have been devised to bias this exploration. These methods range from quite simple to exceedingly complex, and each has its advantages and drawbacks.

The most simple exploration strategy is to take random actions periodically (Luce, 1959; Watkins, 1989). However, these methods do a poor job of reaching states that are difficult to enter and tend to be myopic.

More elaborate methods attempt to build a model of the environment, primarily the p-function, and determine which exploratory actions to take based on this model (Moore & Atkeson, 1993; Wiering & Schmidhuber, 1998). Constructing a model of the environment can be difficult in terms of memory and computation and can be problematic when the environment is dynamic.

Another method of biasing the exploration is augmenting the reward function, r, to alter the behavior of the agent (Matarić, 1994). When maximizing the altered reward signal, the agent may not be improving its true performance metric, just the augmentation of the reward function. So the designer may spend an inordinate amount of time trying to get the reward function "just right", as opposed to having the agent learn the task.

For further analysis of various exploration techniques, the reader is referred to (Whitehead, 1991; Koenig & Simmons, 1993; Lin, 1992; Thrun, 1992).

2.6 Incorporating Prior Knowledge

Incorporating prior knowledge into machine learning tasks has received an increasing amount of attention in recent years. Primarily, knowledge transfer between prior knowledge and a new, unlearned task has dealt with ANNs. Most researchers have focused on transferring the synaptic weights of the ANN used in the prior task and mapping them to help the agent in the new task. The weights from the first ANN typically pass through a task-specific state mapping function that transforms the weights into the state-space of the new task (see Figure 5). This method of incorporating previous knowledge suffers from two serious drawbacks. First, and most critically, is that constructing the state

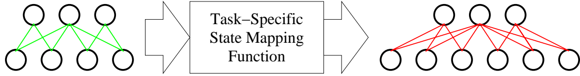

Figure 5: Transferring the synaptic weights of an ANN between tasks.

mapping function is task specific and can be quite difficult. For all but the most trivial tasks, the function will be non-linear and not convex. Second, transferring weights can degrade the performance of the agent if the prior knowledge and mapping function are not chosen properly (Sharkey & Sharkey, 1993). This phenomenon is called "catastrophic interference".

Furthermore, the representation used by most ANNs is not well understood. Therefore, transfer methods tend to be somewhat *ad hoc*.

3 Embedding Prior Knowledge

We propose to embed prior knowledge and previously learned tasks directly into the controller of the reinforcement learning agent. The methods described here are scalable, general, intuitive and can be smoothly integrated with existing reinforcement learning techniques.

3.1 Knowledge Transfer via Guided Exploration

Using an off-policy RL controller scheme (Figure 4), we can embed the prior knowledge into the static controller while learning the value function estimate *tabula rasa*. In essence, we guide the exploration of the RL agent toward an expected interesting area of the state space (i.e., a region of high reward or high knowledge gain). A major benefit of this approach is that any controller that maps states to actions can be used as a prior knowledge source. Neither the representation nor the state space of the prior knowledge controller need be the same as that of the learning controller.

In order for knowledge transfer to occur between the controllers, the learning agent must be capable of observing relevant environment features and able to represent the knowledge internally. For example, if the prior knowledge instructs the agent to recharge its power supply every day at noon, but the RL controller cannot observe the time of day, then the learning agent cannot learn this behavior. We call this a *state-space deficiency*. On the other hand, if the prior knowledge controller uses a history of events to make a decision, a reactive RL agent will not be able to learn this task even if it can observe all of the relevant environmental features. We call this a *representational deficiency*. While the designer must consider the above issues, the overall task is greatly simplified when compared to other approaches (cf. Section 2.6). No longer does the designer have to determine the exact mapping between one knowledge representation and another. With this method, the designer only need determine whether such a mapping exists. Another benefit of this method is that no restriction is placed on the number of prior knowledge sources that can be presented to the agent.

Another benefit of this method is that it allows the agent to utilize more than one prior knowledge source. While it may be possible to create a function that maps information from one representation to another, the task is made even more difficult when multiple sources of information are considered. The method we present allows for the possibility of having multiple knowledge sources guide the exploration of the RL agent. This may involve multiplexing the sources for control of the agent or fusing the action selection distributions of the sources. This allows the learning agent to sample all the actions suggested by the knowledge sources and determine which, if any, is best.

As described in Section 2.4, an off-policy method may cause the RL controller to diverge. This is because the prior knowledge will visit the states according to a different distribution than would the policy that maximizes the current reward function. If this were not the case then the prior knowledge maximizes the reward function and the task is solved anyway. Thus, we slightly augment this scheme in order to bring the state-visitation distributions closer in-line.

3.2 Exploration Control Module

We have augmented the off-policy RL agent to include an exploration control module (Figure 6). The exploration control module selects the actions to be executed in the environment from any number of input sources. The selections made by this module reflect the distributions of all input controllers (i.e., the prior knowledge and RL controllers). Thus, the exploration control module can be designed to have a state-visitation distribution that is arbitrarily similar to that of the RL controller. This alleviates the divergence problems of strict off-policy controllers. Since this method uses an augmented off-policy controller, the body of research developed for improving exploration (discussed in Section 2.5) can be used in the controller. The off-policy controller need not necessarily select the greedy action, but may incorporate the more sophisticated strategies, when the exploration control module selects the actions from off-policy controller.

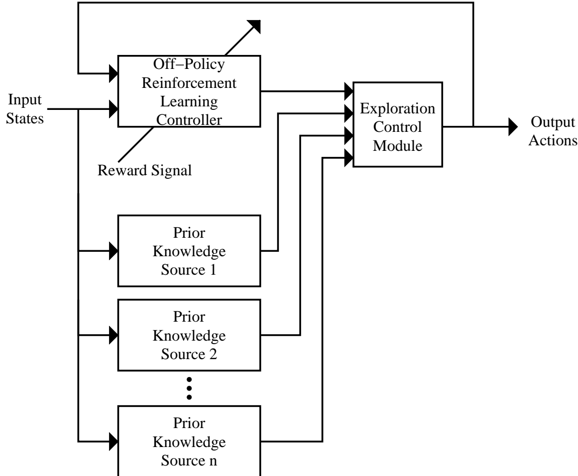

Figure 6: This scheme allows the embedding of prior knowledge directly into the static controller. The selection criterion of the Exploration Control Module can be adjusted to eliminate divergence concerns.

The objectives of the exploration control module are simple: heavily bias the initial exploration toward the actions selected by the prior knowledge and move the state-visitation distribution closer to that of the RL controller as it learns more information. These objectives can be achieved by a variety of methods. Two different implementations of the exploration controller are described in Section 4.

The first method is presented in the context of an incremental learning problem. The agents learn more difficult problems incrementally with the learned controller from one step acting as the prior knowledge to learn the next, more difficult problem. The exploration control module used is a multiplexor that alternates between one of the two controllers for a fixed number of learning steps.

The second exploration controller examined is in the context of composable skill synthesis. A set of source skills is used as prior knowledge. The source skills are tasks related to the target task. A skill is viewed as a reactive mapping of states to actions. The action-selection distribution of each skill, along with the distribution of the RL controller, are linearly combined to form an exploration distribution. The exploration controller then selects actions from this distribution.

4 Applications

In this section, we demonstrate that our approach for embedding prior knowledge can be used to improve the performance and decrease the learning time of RL agents. One example is taken from the simulated mobile-robot domain, the other from the grid-world arena. In Section 4.1, we use our approach to solve a complex task incrementally and, in Section 4.2, to utilize previously learned skills to compose new skills.

4.1 Incremental Learning

In incremental learning, a large, complex task is decomposed into smaller sub-tasks. If the task is decomposed properly then solving all the sub-tasks may be *easier* than solving the entire task, and by solving the sub-tasks, a solution to the target task is found. By easier, we mean that some objective measure of performance is greater than (or less than, as the case may be) the same objective measure resulting from using another methodology.

4.1.1 Description of Problem

To demonstrate our approach in the incremental learning domain, we selected a game of mobile robot “tag”. In this game, there three mobile robots: two *Runners* and one *Defender*. The Runners attempt to move to a goal location and the Defender attempts to stop either Runner from reaching the goal. The game is pictured in Figure 7. The task is to find a policy for both Runners that maximizes its expected sum of future rewards. The policy for the Defender is fixed and does not change during the course of this experiment. The reward function for the Runners is +1 for reaching the goal, -1 for being tagged (i.e., within one meter of the Defender) or moving too far away (greater than ten meters) from the goal, and zero otherwise. A game begins with the Runners randomly distributed around a disc ten meters away from the goal and the Defender in the goal location. A game ends when either Runner enters the goal location or the Defender tags one of the Runners. All robots are restarted and the process is repeated. The reward function is not shared between the Runners; a Runner only receives a reward when it scores, is tagged, or moves too far from the goal, and receives no credit for what the other Runner does. Thus, there is no explicit attempt to encourage the Runners to collaborate. Collaboration must be learned through attempts to maximize the individual reward function of the Runner. The control software was written using the Port-Based Adaptable Agent Architecture (Dixon et al., 2000) and the simulations were executed using the Real and Virtual Environment engine (Dixon et al., 1999).

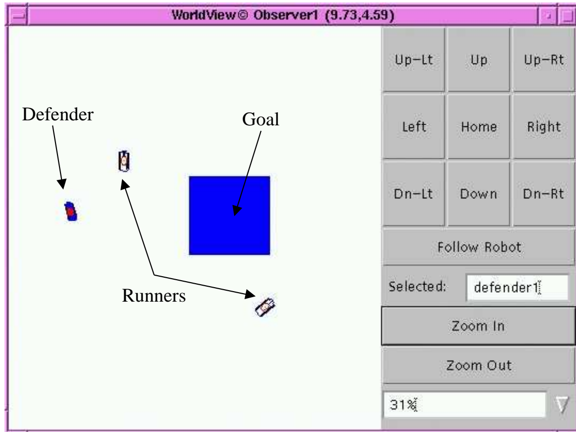

Figure 7: The tag game used to demonstrate incremental learning.

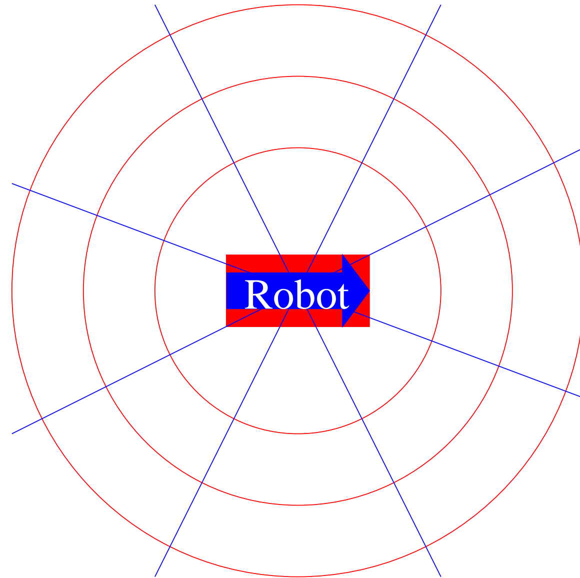

Figure 8: The states for the Runners: distances were discretized into eight concentric circles and bearings into eight pie-wedges.

4.1.2 Experimental Setup

To compare the performance of the incremental method versus *tabula rasa*, we decided that both sets of Runners would learn for 4,000 games. For the incremental method, the task was manually decomposed into three steps. First, one Runner plays 100 games by itself (no Defenders). Basically, the Runner will learn to move directly for the goal. Next, the Runner uses the ability to go to the goal as the previously learned information in a series of 1,900 games where one Runner plays against one Defender. This increment allows the robot to learn how to score when a Defender is attempting to tag it. Finally, two Runners use this as the previously learned information for 2,000 games against one Defender.

The simulated robots were equipped with a omni-directional camera that allows the robot to determine the relative distance and bearing to the Runners, Defender, and goal (orientation and velocity of those objects cannot be determined without further processing of the sensor information and was disregarded). The distances were discretized into eight concentric circles and the bearings were discretized into eight pie-wedges (Figure 8) and each "bin" in the state-space is assigned a unique number. Using this scheme requires 2^6 states to represent the distance and bearing of a single object. Since there are three objects in the game (the goal and the two other robots), there are $2^{18} = 262,144$ states in the system. Q-Learning was selected as the RL algorithm and a single-layer perceptron was chosen to represent the Q-function.

It is important to note that all the above decisions were made prior to any results from these methods. Four-thousand games seemed like a reasonable number of games to allow the Runners to develop a sufficiently good strategy for scoring. Similarly, the number of games at each increment (100 then 1,900 then 2,000) seemed like a reasonable number of games to learn the sub-task and was chosen before results were obtained.

4.1.3 Experimental Results

After both the incremental and *tabula rasa* controllers had learned for 4,000 games, we compute performance metrics by allowing the Runners play against the Defender for 2,500 games. Each game is viewed as a Bernoulli trial, and the probability of scoring from the Bernoulli trial is viewed as the performance metric. The probability of scoring for the *tabula rasa* Runners was 0.2916 while the incremental learning Runners had a 0.5724 probability of scoring. This is almost a 100 percent improvement in performance. The error bars on Figure 9 indicate the 95 percent confidence intervals on the probability of scoring. Furthermore, the *tabula rasa* Runners took 9.6 simulation days to learn during the 4,000 games, while the incremental learning Runners took 3.6 simulation days, as shown in Figure 10. This is an improvement of 166 percent. (A simulation day is equivalent to the number of days of execution in real time, multiplied by the simulation rate. Thus, a simulation that took one day of real time to complete, and the simulation rate was ten to one, would be equivalent to ten simulation days.)

From these results, it is clear that incremental learning using our method can drastically improve the per-

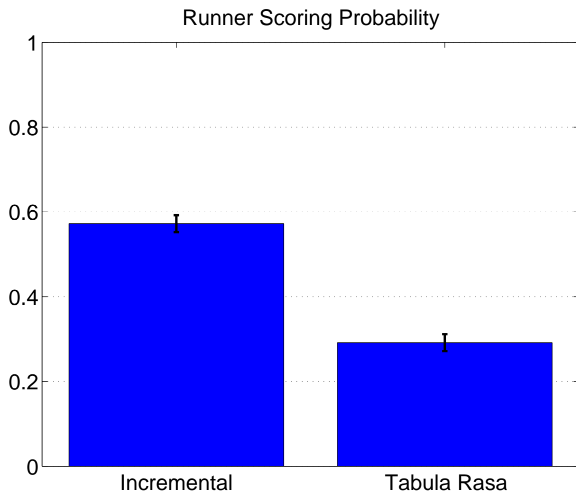

Figure 9: The probability of scoring for the incremental learning and *tabula rasa* Runners. The error bars indicate the 95 percent confidence intervals for this probability parameter.

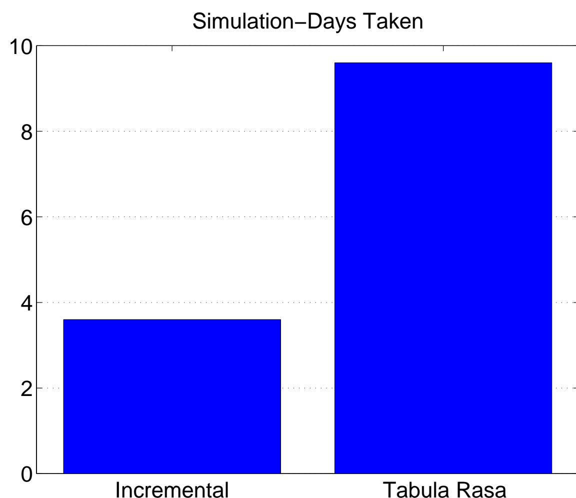

Figure 10: The number of simulation days required to play the 4,000 games in the learning phase.

formance and decrease the time required to learn a task.

4.2 Composable Skill Synthesis

In composable skill synthesis, a problem is broken down into a set of basic skills that the agent must possess in order to complete the task. Using the knowledge of these basic skills, the agent may learn the more complex target problem more quickly than if there was no prior skill knowledge. The motivation of this work is to encourage the reuse of previously acquired knowledge and promote the composablility of autonomous systems. In the context of this work, a skill is defined as a reactive mapping of states to actions. Hence, any policy learned by a RL agent qualifies as a skill. Additionally, this mapping policy need not be deterministic.

In the incremental learning problem of the previous section, the policy from a particular learning increment can be considered a skill that is provided to the agent learning the next, more difficult problem. The contrast between the work in this section and the incremental learning work is that this section assumes the pre-existence of some set of skills from which the agent can draw knowledge. Now, the task of the designer becomes the selection of appropriately related skills from this knowledge base and the incorporation of the skills into the learning process. This creates the need to integrate the knowledge from any number of source skills into the learning process.

4.2.1 Exploration Control for Multiple Knowledge Sources

The optimal method to integrate knowledge from multiple sources is not clear. In some cases, a temporally controlled multiplexor, such as that in Section 4.1, may work well. Note that the multiplexor method could be extended to incorporate multiple knowledge sources. However, in some situations that approach may not be appropriate. Consider the example of a mobile robot learning to navigate through a cluttered office building to deliver mail. The prior skills might be avoidance of static objects, avoidance of dynamic objects, and goal homing. It is difficult to know *a priori* when each skill would be needed in a dynamic environment. Thus, this would be a situation where the agent should not be constrained to use one skill for some predetermined length of time. Also, it is highly desirable for the agent to incorporate newly acquired information as soon as it becomes available.

We propose that knowledge from multiple sources can be combined in the form of weighted action selection probability distributions. Thus, the agent can con-

sider all skills simultaneously and newly learned information can be promptly incorporated into the control of the agent.

Define $p(a|s)$ as the probability of selecting action a given the agent is in state s. Thus, $p_i(\cdot|s)$ is the action selection distribution for skill $i \in \{1, \ldots, N\}$, where N is the number of source skills. Similarly, we define $p_0^t(\cdot|s)$ as the distribution for the target skill at time t. Let w_j be the weight applied to distribution $j \in \{0, \ldots, N\}$. Defining the vectors

$$\mathbf{p}^t \triangleq [p_0^t(\cdot|s), p_1^t(\cdot|s), \ldots, p_N^t(\cdot|s)]^T$$

$$\mathbf{w} \triangleq [w_0, w_1, \ldots, w_N]^T$$

and assuming all weights are greater than or equal to zero, we can state the distribution of the exploration controller module as

$$p_{ECM}(\cdot|s) = \frac{1}{\sum_{i=0}^{N} w_i} \mathbf{w}^T \mathbf{p}^t$$

Actions to be executed in the environment are selected from p_{ECM}. One approach is to select actions directly based upon the distribution. A second approach, called ϵ-greedy, is to select the greedy action (i.e., the action with the highest Q-value) from this distribution with a probability of $(N - \epsilon(N-1))/N$ and to select a non-greedy action with uniform distribution otherwise.

Clearly, both ϵ-greedy selection and direct selection from p_{ECM} have benefits and drawbacks which must be considered during the design of the agent.

4.2.2 Experimental Setup

This section describes the experiments conducted using the experimental controller described in the previous section. The task is to navigate the grid world shown in Figure 11. The agent must navigate from the start position, marked with an *S*, to the goal position, marked with a *G* without running into any objects (shaded areas).

While learning, a reward of +1 is received by the agent for entering the goal position and a reward of -1 for striking any of the walls. A reward of 0 is received after all other moves. Grid world sizes of 10 by 10 and 25 by 25 were used. The learning task is simple but proves sufficient to demonstrate knowledge transfer.

The state of the agent is composed of the coordinates of the grid square it occupies and its orientation. The agent has eight possible orientations corresponding to the primary and secondary compass directions (Figure 12). This means that for a 10 by 10 grid world, the agent has $10 * 10 * 8 = 800$ possible states. For

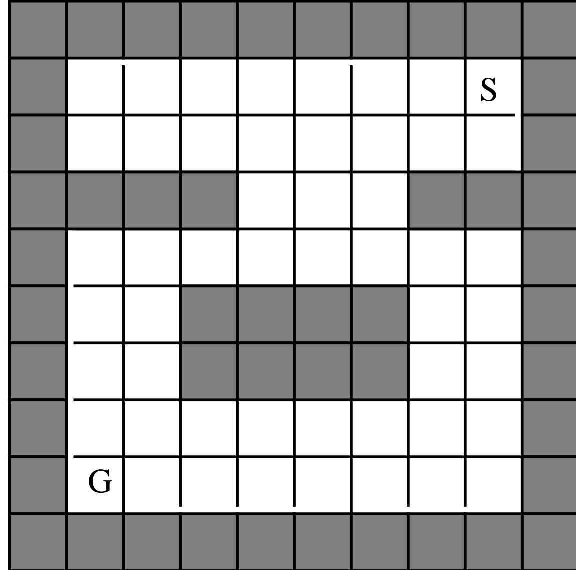

Figure 11: The grid world used in the skill transfer experiments. The agent starts at S and navigates to the goal point G.

a 25 by 25 grid world the number of states is 5000. The actions available to the agent are to move *forward* (FWD), turn *clockwise* (CW) one orientation position or turn *counter-clockwise* (CCW) one orientation position.

One must be conscientious of the differences between this state space and other grid-world state spaces. In this construction, the grid position directly behind the agent is not a neighboring state. While it is a neighboring grid position, it takes a sequence of five actions to reach this position (a sequence of four turns to the same direction followed by one forward action). Thus, the grid positions and the system states do not have as obvious of a relationship as they do in some other grid-world examples. This type of representation was chosen to better reflect the behavior of an actual robot, which typically has a distinct front that faces the direction of translation.

The experiments compared relative learning rates of two types of agents. One agent is supplemented with skill knowledge (the skill-based agent or SBA) and the other is not (the *tabula rasa* agent or TRA). In all other respects, the agents are the identical. Both agents utilize tabular Q-learning methods. The tabular version of the algorithm does not perform generalization and was chosen for its simplicity and ease of analysis. To speed the learning process, both agents we subjected to experience replay (Lin, 1992). In experience replay, the agent updates its parameter estimates according to its previous episode, played in reverse order. The agent still up-

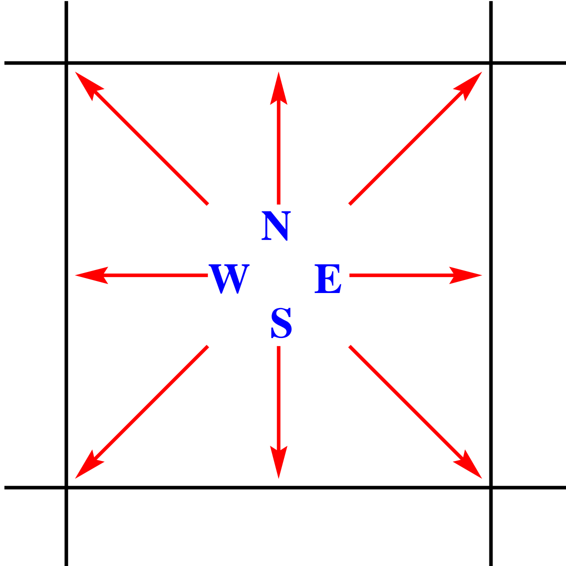

Figure 12: Possible agent orientations within the grid world (N, NE, E, SE, S, SW, W, NW).

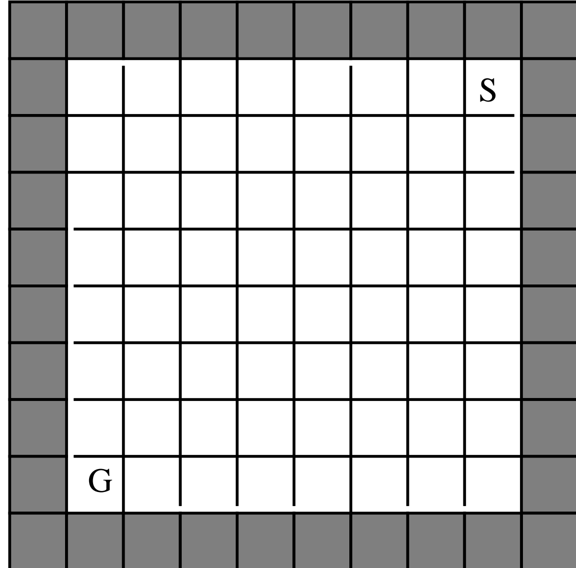

Figure 13: grid-world used to learn the *goal homing* skill.

dates its estimates during the episode. In the work that Lin performed, only the actions that corresponded to the current policy were used in the experience replay. We utilize tabular Q-learning and hence are able to replay the entire episode.

The skills used in the SBA are object avoidance and goal homing. Object avoidance (OA) is a hand-written skill that directs the agent away from objects. This skill assumes that the agent possess a sensor that can view the eight neighboring grid positions and determine whether or not there is an obstacle present. If there is no object directly in front of the agent, the output of the OA skill is a deterministic policy giving probability one to the FWD action and probability zero to the turn actions (CW and CCW). If there is an object directly in front of the agent, the skill gives probability zero to the FWD action. The skill then identifies the non-occupied grid position nearest to forward direction from among the seven remaining neighbor cells. The skill policy is to turn in the direction of the unoccupied cell. In the event of a tie each of the turns (CW and CCW) are given equal probabilities (of one-half).

The state information utilized by the OA skill is not directly fed into the learning controller. However, the learning controller would eventually develop an equivalent to this information in its own representation whether the object avoidance skill is provided *a priori* or not. The primary difference between the OA skill and the representation that the learning controller will acquire is that the OA skill is more general. The learning controller will obtain a representation of the exact grid world presented. In a sense, it will learn a map of the object locations in this particular world. The OA skill is more general and is representative of a skill that we might keep in a skill-base.

The goal homing (GH) skill, unlike the OA skill, is learned. It is the result of learning a simplified grid-world that is partially related to the target learning task (Figure 13). The agent learns to reach the goal position from the start position, but there are no objects in the course (except the outer walls). The skill was trained until the agent converged upon the optimal policy from the start position given. It is not necessarily optimal from all points in the state space. The policy produced by the GH skill is computed using a Boltzman distribution such that

$$p(a|s) = \frac{e^{Q(s,a)}}{\sum_{a' \in A} e^{Q(s,a')}}$$

is the probability of taking an action, a, in a particular state, s.

4.2.3 Experimental Results

The results of the experiments show the SBA outperforming the TRA. We assume here that some cost is associated with each action taken in the environment during learning (perhaps computing expenses or execution time). Thus, the number of actions needed to reach the

Figure 14: Reward earned for a test run versus the number of learning steps taken for the 10 by 10 grid-world. The skill-based agent clearly outperforms the *tabula rasa* agent. The results are an average of 30 different experiments for each agent type.

goal is used as a performance metric. When counting actions taken by the agent, we often use the term *steps*.

Figure 14 shows the relative performance of the two learning agents. Every 250 learning steps, the current greedy policy of the RL controller is evaluated by recording the number of steps needed to reach the goal state. Evaluation ties for the greedy action are resolved stochastically. At the time of action selection, one action is chosen with uniform distribution over the set of actions that the controller has deemed equally good. This means that the agent may strike many walls before reaching the goal and initially performs what is essentially a random walk.

The SBA converges to the optimal solution in less than 2,500 learning steps (see Figure 14). The TRA eventually obtains the same solution, but it takes nearly 47,500 steps. The SBA is faster by nearly a factor of 20 in a world with only 1,224 reachable state-action pairs. (Grid positions that contain objects are unreachable by the agent and result in several unreachable state-action pairs.)

The difference is even more dramatic in the 25 by 25 tests. This world has 9,744 reachable state-action pairs making this world almost eight times as complex as the 10 by 10 case. Over the course of 20 different experiments, the SBA converges to an optimal solution in an average of less than 4,000 steps while the TRA was found to take well over 300,000 steps on the average. The SBA is faster by a factor of more than 75.

5 Related Work

Much research has been conducted in the area of exploration for reinforcement learning agents. Whitehead (Whitehead, 1991) showed that random exploration in a deterministic world was yields a learning time that is exponential in the number of states. Thrun (Thrun, 1992) showed that directed exploration can learn in time that is polynomial in the number of states for deterministic environments. Directed exploration methods do not explore based on randomness, but instead use other information, such as state visitation counts, to guide the search. Note that directed exploration may cause an off-policy controller (Figure 4) to diverge. In the same paper, Thrun also proves that learning time can be exponential in stochastic environments no matter what exploration method is used. The role of the state-space representation has on exploration and learning complexity has been studied in (Koenig & Simmons, 1993).

There has been some prior work in the area of incorporating prior knowledge into a reinforcement learning agent. Lin (Lin, 1992) addresses the issue of teaching a RL agent. In his work, a teacher provides the agent with a “lesson”. A lesson is a demonstration of how to achieve the target task from a given initial state. The learning information from this demonstrated episode is played back for the RL agent which, in turn, updates its utility estimates. This work is limited to using teachers that know how to accomplish the entire task. A teacher that is only sufficient in certain aspects of the task may not be of any benefit to the learning agent. The work also does not address the issue of multiple teachers.

Maclin and Shavlik (Maclin & Shavlik, 1996) have created a system that incorporates user-given advice into a reinforcement learning agent. The user provides advice by issuing commands in an imperative programming language. This advice is converted from rules to weights for an artificial neural network (ANN) using knowledge-based ANN (KBANN) methods and is directly installed into the ANN that estimates the value function. Their methods were empirically found to perform well. In another study, (Maclin, 1995) demonstrates an ability to overcome “bad” advice. However, their approach is complex, involving a specialized programming language and KBANNs which are often have several hidden layers.

Whitehead (Whitehead, 1991) also studies methods for incorporating prior knowledge into the reinforcement learning process. He describes two approaches: learning with an external critic (LEC) and learning by watching (LBW). In LEC, a critic provides feedback to the agent in supplement to the true environmental reward signal. The LBW approach is similar to that used

by Lin.

6 Conclusions and Future Work

We have proposed an intuitive method for embedding previously learned information and prior knowledge into the controller of a reinforcement learning agent. This method promises drastic improvements in the performance of the RL agent, reductions in learning time, and utilizes a representation that eases analysis and increases intuition into the problem. Furthermore, we have demonstrated that this method can be applied to several types of relevant problems.

The results of the incremental learning section (Section 4.1) demonstrate the feasibility of this approach to improve the performance and reduce the learning time of complex tasks significantly. However, the target task was decomposed manually and each increment was chosen because it appeared to be relevant to accomplish the target task. The performance of the incremental learning method is very sensitive to the increments selected. If the increments were chosen in a more Byzantine fashion, then the results would decline (comparision experiments have been completed but not enough data have been collected at this time). However, the primary difference in results is in the learning time, not in the performance of the system itself.

Future work in this area will involve the automation of the decomposition of the target task. Though not mentioned in the results portion (Section 4.1.3 of the incremental learning section, a considerable amount of time was spent manually selecting and designing the increments for the target task. This automation will decrease the amount of human intervention in the learning process, and would hopefully increase the overall time required to learn a task.

The results of the composable skills section (Section 4.2) demonstrate that it is feasible to incorporate knowledge from multiple skill sources into a reinforcement learning agent. The rate of learning was significantly improved when prior skill knowledge was embedded in the agent. The decrease in learning time, relative to a *tabula rasa* system, was found to be greater as the state space size was increased.

It is clear that the skills utilized in this experiment (goal homing and object avoidance) are not both applicable in all states of the system. It may be more appropriate to have a weighting function that varies over the state space. However, such a variation in the weighting function would be difficult to design. If the designer knew the best weighting function beforehand, the problem of composing the skills is already solved. Thus, it becomes useful for the agent to adjust its weights as it learns more about its environment. The way in which the agent modifies its weights and the way in which it determines which weights to modify are important issues. Future work is planned in this area.

The skills utilized were also highly related to the target skill (finding the goal without collisions). This high relatedness leads to a large improvement in learning rate. If inappropriate skills are selected, it is doubtful that the agent will outperform a *tabula rasa* system by such a large margin. Thus, either the designer must always make good decisions about which skills to include or the agent must be able to disregard skills that turn out to be poor selections. This behavior, the ability to ignore bad advice, would be important in an autonomous agent. The problem is one of lowering the weight of a source skill that provides bad action selections and is therefore related to the above problem of varying weights in general.

It is not clear from this study how each skill effected the learning process. We feel that the inclusion of either of the skills alone would have dramatically increased the speed of learning. The inclusion of a second skill may have only offered a small increase beyond that. The effects of each skill must be better understood before any weight update rule can be derived and before clear design guidelines can be laid out.

Acknowledgments

We would like to thank Enrique Ferreira, Jonathan Jackson, and Bruce Krogh for their contributions to this work.

This work was supported in part by DARPA/ETO under contract F30602-96-2-0240 and by the Institute for Complex Engineered Systems at Carnegie Mellon University. We also thank the Intel Corporation for providing the computing hardware.

References

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 30–37).

Baxter, J. (1994). *Learning internal representations*. Doctoral dissertation, The Flinders University of South Australia.

Bellman, R. E. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems* (pp. 369–376).

Bradtke, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. *Advances in Neural Information Processing Systems* (pp. 295–302).

Dixon, K. R., Dolan, J. M., Huang, W., Paredis, C. J., & Khosla, P. K. (1999). RAVE: A real and virtual environment for multiple mobile robot systems. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Dixon, K. R., Pham, T. Q., & Khosla, P. K. (2000). Port-based adaptable agent architecture. *International Workshop on Self-Adaptive Software*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Koenig, S., & Simmons, R. G. (1993). Complexity analysis of real-time reinforcement learning. *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI).* (pp. 99–105).

Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 294–321.

Luce, D. (1959). *Individual choice behavior*. New York, NY: John Wiley.

Maclin, R. (1995). *Learning from instruction and experience: Methods for incorporating proceedural domain theories into knowledge-based neural networks.* Doctoral dissertation, Computer Sciences Department, University of Wisconsin, Madison.

Maclin, R., & Shavlik, J. (1996). Creating advice-taking reinforcement learners. *Machine Learning*, *22*, 251–282.

Matarić, M. J. (1994). Reward functions for accelerated learning. *Proceedings of the International Conference on Machine Learning* (pp. 181–189).

Moore, A. W., & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, *13*, 103–130.

Pratt, L. (1994). *Experiments on the transfer of knowledge between neural networks*. Cambridge, MA: MIT Press.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*, 210–229.

Sharkey, N. E., & Sharkey, A. J. (1993). Adaptive generalisation and the transfer of knowledge. *AI Review*, *7*, 313–328.

Tesauro, G., & Sejnowski, T. J. (1992). A parallel network that learns to play backgammon. *Artificial Intelligence*, *39*, 357–390.

Thrun, S. B. (1992). *Efficient exploration in reinforcement learning* (Technical Report CMU-CS-92-102). Carnegie Mellon University, Pittsburgh, PA.

Tsitsiklis, J. N., & Van Roy, B. (1995). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Watkins, C. J. (1989). *Learning with delayed rewards*. Doctoral dissertation, King's College, Cambridge University.

Whitehead, S. (1991). *A study of cooperative mechanisms for faster reinforcement learning* (Technical Report 365). University of Rochester, Rochester, NY.

Wiering, M., & Schmidhuber, J. (1998). Efficient model-based exploration. *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*.