Speeding Up Reinforcement Learning with Behavior Transfer

Matthew E. Taylor and Peter Stone
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188
{mtaylor, pstone}@cs.utexas.edu

Abstract

Reinforcement learning (RL) methods (Sutton & Barto 1998) have become popular machine learning techniques in recent years. RL has had some experimental successes and has been shown to exhibit some desirable properties in theory, but it has often been found very slow in practice. In this paper we introduce *behavior transfer*, a novel approach to speeding up traditional RL. We present experimental results showing a learner is able learn one task and then use behavior transfer to markedly reduce the total training time for a more complex task.

Introduction

Reinforcement learning (Sutton & Barto 1998) has shown some success in different machine learning tasks because of its ability to learn where there is limited prior knowledge and minimal environmental feedback. However, reinforcement learning often is very slow to produce near-optimal behaviors. Many techniques exist which attempt, with more or less success, to speed up the learning process.

Past research (Selfridge, Sutton, & Barto 1985) has shown that a learner can train faster on a task if it has first learned on a simpler variation of the task, referred to as *directed training*. In this paradigm the state transition function, which is part of the environment, can change between tasks. *Learning from easy missions* (Asada *et al.* 1994) is a technique that relies on human input to modify the starting state of the learner over time, making it incrementally more difficult for the learner. Both of these methods reduce the total training time required to successfully learn the final task. However, neither allow for changes to the state or action spaces between the tasks, limiting their applicability. *Reward shaping* (Colombetti & Dorigo 1993; Mataric 1994) allows one to bias a learner's progress through the state space by adding in artificial rewards to the environmental rewards. Doing so requires sufficient knowledge about the environment a priori to guide the learner and must be done carefully to ensure that unintended behaviors are not introduced. While it is well understood how to add this type of guidance to a learner (Ng, Harada, & Russell 1999), we would prefer to allow the agent to learn faster by training on different (perhaps pre-existing) tasks. Using behavior transfer we are able to leverage existing learned knowledge as well as speed up tasks in domains where existing schemes to accelerate reinforcement learning are not applicable.

In this paper we introduce *behavior transfer*, whereby a learner trained on one task can learn faster when training on another task with related, but different, state and action spaces. Behavior transfer is more general than the previously referenced methods because it does not preclude the modification of the transition function, start state, or reward function. The key technical challenge is mapping a value function in one representation to a meaningful value function in another, typically larger, representation. The primary contribution of this paper is to establish an existence proof that there are domains in which it is possible to construct such a mapping and thereby speed up learning via behavior transfer.

Behavior Transfer Methodology

To formally define behavior transfer we first briefly review the general reinforcement learning framework that conforms to the generally accepted notation for Markov decision processes (Puterman 1994). There is some set of possible perceptions of the current state of the world, S, and a learner has some initial starting state, $s_{initial}$. When in a particular state s there is a set of actions A which can be taken. The reward function R maps each perceived state of the environment to a single number which is the value, or instantaneous reward, of the state. T, the transition function, takes a state and an action and returns the state of the environment after the action is performed. If transitions are non-deterministic the transition function is a probability distribution function. A learner is able to sense s, and typically knows A, but may or may not initially know R or T.

A policy $\pi : S \mapsto A$ defines how a learner interacts with the environment by mapping perceived environmental states to actions. π is modified by the learner over time to improve performance, i.e. the expected total reward accumulated, and it completely defines the behavior of the learner in an environment. In the general case the policy can be stochastic. The success of an agent is determined by how well it maximizes the total reward it receives in the long run while acting under some policy π. An $optimal\ policy$, π^*, is a policy which does maximize this value (in expectation). Any reasonable learning algorithm attempts to modify

π over time so that it reaches π^* in the limit.

Past research confirms that if two tasks are closely related the learned policy from one task can be used to provide a good initial policy for the second task. For example, Selfridge (1985) showed that the 1-D pole balancing task could be made harder over time by shortening the length of the pole and decreasing its mass; when the learner was first trained on a longer and lighter pole it could more quickly learn to succeed in the more difficult task with the modified transition function. In this way, the learner is able to refine an initial policy for a given task: $(S_1, s_{(1,initial)}, A_1, T_1, R_1, \pi_0) \Rightarrow \pi_{(1,final)}$ where task 1 starts from no initial policy as indicated by the π_0 in the last value of the tuple. Task 2 can then be defined as $(S_2, s_{(2,initial)}, A_2, T_2, R_2, \pi_{(2,initial)}) \Rightarrow \pi_{(2,final)}$. The time it takes to learn $\pi_{(2,final)} = \pi_2^*$ may be less for $(S_2, s_{(2,initial)}, A_2, T_2, R_2, \pi_{(1,final)})$ than $(S_2, s_{(2,initial)}, A_2, T_2, R_2, \pi_0)$. Note that since $S_1 = S_2$ and $A_1 = A_2$, $\pi_{(1,final)}$ is a legitimate policy for task 2.

In this paper we consider the more general case where $S_1 \neq S_2$, and/or $A_1 \neq A_2$. To use the policy $\pi_{(1,final)}$ as the initial policy for the second task, we must transform its value function so that it can be directly applied to the new state and action space. A behavior transfer function $\rho(\pi)$ will allow us to apply a policy in a new task $(S_2, s_{(2,initial)}, A_2, T_2, R_2, \rho(\pi_{(1,final)})) \Rightarrow \pi_{(2,final)}$. The policy transform function ρ needs to modify the policy so that it accepts the states in the new task as inputs and allows for the actions in the new task to be outputs. A policy generally selects the action which is expected to accumulate the largest expected total reward and thus the problem of transforming a policy between two tasks reduces to transforming the value function. Defining ρ to do this correctly is the key technical challenge to enable general behavior transfer.

One measure of success in speeding up learning using this method is that given a policy π_1, the training time for $\pi_{(2,final)}$ to reach some performance threshold decreases when replacing the initial policy π_0 with $\rho(\pi_1)$. Let $time(S, s_{initial}, A, T, R, \pi)$ be the time it takes to find a near-optimal policy in the task. If behavior transfer works, $time(S_2, s_{(2,initial)}, A_2, T_2, R_2, \rho(\pi_{(1,final)})) < time(S_2, s_{(2,initial)}, A_2, T_2, R_2, \pi_0)$. This criterion is relevant when task 1 is given and is of interest in its own right.

A stronger measure of success is that the training time for both tasks using behavior transfer is shorter than the training time to learn the second task from scratch. In other words, $time(S_1, s_{(1,initial)}, A_1, T_1, R_1, \pi_0^{'}) + time(S_2, s_{(2,initial)}, A_2, T_2, R_2, \rho(\pi_{(1,final)})) < time(S_2, s_{(2,initial)}, A_2, T_2, R_2, \pi_0)$. This criterion is relevant when task 1 is created for the sole purpose of speeding up learning via behavior transfer.

Testbed Domain

To demonstrate the effectiveness and applicability of the behavior transfer method (detailed in section 5) we empirically test it in the RoboCup simulated soccer keepaway domain using a setup similar to past research (Stone & Sutton 2002; Kuhlmann & Stone 2004). RoboCup simulated soccer is well understood as it has been the basis of multiple international competitions and research challenges. The multiagent domain incorporates noisy sensors and actuators, as well as enforcing a hidden state so that agents can only have a partial world view at any given time. While there has been previous work which attempted to use machine learning to learn the full simulated soccer problem (Andre & Teller 1999; Riedmiller *et al.* 2001), the complexity and size of the problem have so far proven prohibitive. However, many of the RoboCup subproblems have been isolated and solved using machine learning techniques, including the task of playing keepaway.

Keepaway, a subproblem of RoboCup soccer, is the challenge where one team, the *keepers*, attempts to maintain possession of the ball on a field while another team, the *takers*, attempts to gain possession of the ball or force the ball out of bounds, ending an *episode*. Keepers that are able to make better decisions about their actions are able to maintain possession of the ball longer and thus have have a longer average episode length. Figure 1 depicts three keepers playing against two takers. [1]

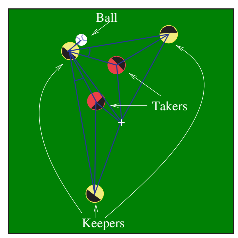

Figure 1: This diagram depicts the 13 state variables used for learning with 3 keepers and 2 takers. There are 11 distances to players, the center of the field, and the ball, as well as 2 angles along passing lanes.

As more players are added to the task, keepaway becomes harder for the keepers because the field becomes more crowded. As more takers are added there are more players to block passing lanes and chase down any errant passes. As more keepers are added, the keeper with the ball has more passing options but the average pass distance is shorter. This forces more passes and will lead to more errors because of the noisy actuators and imperfect perception. For this reason keepers in 4 vs. 3 keepaway (meaning 4 keepers and 3 takers) take longer to learn an optimal control policy than in 3 vs. 2. The hold time of the best policy for a constant field size will also decrease when moving from 3 vs. 2 to 4 vs. 3 due to the added difficulty. This has been discussed in previous research (Kuhlmann & Stone 2004).

The different keepaway tasks are all problems which may occur during a real game. Learning on one task and transferring the behavior to a separate useful task can reduce the training time. In the keepaway domain, A and S are determined by the current keepaway task and thus differ from instance to instance. However, $s_{initial}$, R and T, though formally different, are effectively constant across tasks. When S and A change, $s_{initial}$, R, and T change by definition. But in practice, R is always defined as 1 for every time step that the keepers maintain possession, and $s_{initial}$ and T are always defined by the RoboCup soccer simulation.

[1] Flash-file demonstrations of the task can be found at *http://www.cs.utexas.edu/users/AustinVilla/sim/keepaway/*.

Learning Keepaway

The keepers use episodic SMDP Sarsa(λ) (Sutton & Barto 1998) to learn their task. We use linear tile-coding function approximation, also known as CMACs, which has been successfully used in many reinforcement learning systems (Albus 1981). The keepers choose not from primitive actions (turn, dash, or kick) but higher-level actions implemented by the CMUnited-99 team (Stone, Riley, & Veloso 2000). A keeper without the ball automatically attempts to move to an open area (the receive action). A keeper in possession of the ball has the freedom to decide whether to hold the ball or to pass to a teammate.

Function approximation is often needed in reinforcement learning so that the learner is capable of generalizing the policy to perform well on unvisited states. CMACs allow us to take arbitrary groups of continuous state variables and lay infinite, axis-parallel tilings over them (see Figure 2). Using this method we are able to discretize the continuous state space by using tilings while maintaining the capability to generalize via multiple overlapping tilings. The number of tiles and width of the tilings are hardcoded and this dictates which state values will activate which tiles. The function approximation is learned by changing how much each tile contributes to the output of the function approximator. By default, all the CMAC's weights are initialized to zero. This approach to function approximation in the RoboCup soccer domain is detailed by Stone and Sutton (2002).

For the purposes of this paper, it is important to note the state variables and action possibilities used by the learners. The keepers' states comprise distances and angles of the keepers $K_1 - K_n$, the takers $T_1 - T_m$, and the center of the playing region C (see Figure 1). Keepers and takers are ordered by increasing distance from the ball. Note that as the number of keepers n and the number of takers m increase, the number of state variables also increase so that the more complex state can be fully described. S must change (e.g. there are more distances to players to account for) and $|A|$ increases as there are more teammates for the keeper with possession of the ball to pass to. Full details of the keepaway domain and player implementation are documented elsewhere (Stone & Sutton 2002).

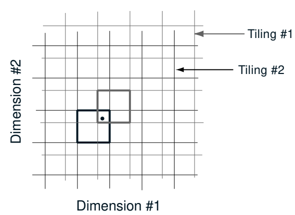

Figure 2: The tile-coding feature sets are formed from multiple overlapping tilings. The state variables are used to determine the activated tile in each of the different tilings. Every activated tile then contributes a weighted value to the total output of the CMAC for the given state. Note that we primarily use one-dimensional tilings but that the principles apply in the n-dimensional case.

Learning 3 vs. 2

On a 25m x 25m field, three keepers are initially placed in the three corners of the field and a ball is placed near one of the keepers. The two takers are placed in the fourth corner. When the episode starts, the three keepers attempt to keep control of the ball by passing amongst themselves and moving to open positions. The keeper with the ball has the option to either pass the ball to one of its two teammates or to hold the ball. We allow the keepers to learn to choose between these three choices when in control of the ball. In this task A = {hold, passToTeammate1, passToTeammate2}. S is defined by 13 state variables, as shown in Figure 1. When a taker gains control of the ball or the ball is kicked out of the field's bounds the episode is finished. The reward to the Sarsa(λ) algorithm for the keeper is the number of time steps the ball remains in play after an action is taken. The episode is then reset with a random keeper placed near the ball.

All weights in the CMAC function approximator are initially set to zero and therefore $\pi_{(3vs2,initial)} = \pi_0$. As training progresses, the weight values are changed by Sarsa(λ) so that the average hold time of the keepers increases. Throughout this process, the takers use a static hand-coded policy to attempt to capture the ball as quickly as possible. Due to the large amounts of randomness in the environment, the evaluation of a policy is very noisy.

Learning 4 vs. 3

Holding the field size constant we now add an additional keeper and an additional taker. R and T are essentially unchanged from 3 vs. 2 keepaway, but now A = {hold, passToTeammate1, passToTeammate2, passToTeammate3} and S is made up of 19 state variables due to the added players. The 4 vs. 3 task is harder than the 3 vs. 2 task and the learned average hold times after 20 hours of training from $\pi_{initial} = \pi_0$ decrease from roughly 13.6 seconds for 3 vs. 2 to 9.3 seconds for 4 vs. 3.

In order to quantify how fast an agent in 4 vs. 3 learns, we set a threshold of 9.0 seconds. When a group of four keepers has learned to hold the ball from the three takers for an average of 9.0 seconds over 1,000 episodes we say that the keepers have sufficiently learned the 4 vs. 3 task. By recording this time over many trials we can measure the effectiveness of the Sarsa(λ) algorithm in different situations.

Behavior Transfer in Keepaway

To define a ρ which will correctly transfer behavior from $\pi_{(3vs2,final)}$ into $\pi_{(4vs3,initial)}$, the value function utilized by π needs to handle the new state and action spaces reasonably. In the keepaway domain we are able to intuit the mappings between actions in the two tasks and states in the two tasks based on our knowledge of the domain. Our choice for the mappings is supported by empirical evidence showing that behavior transfer decreases training time. Other domains will not necessarily have such straightforward transforms between tasks of different complexity. Finding a general method to specify ρ is outside the scope of this paper and will be formulated in future work. One of the main challenges will be identifying general heuristics for mapping existing states and actions in the first task to new states and actions in a second task. Creating a general metric for similarity between state variables and actions in two tasks would allow us to identify a promising mapping for rho and give an a priori indication of whether behavior transfer will work in a particular domain. Our primary contribution in this paper is demonstrating that there exist domains in which ρ can be constructed and then used to successfully increase the learning rate.

The naive approach of directly copying the CMAC's weights to duplicate the value function from $\pi_{(3vs2,final)}$ into $\pi_{(4vs3,initial)}$ fails because both S and A have changed. Keeping in mind that $\pi : S \mapsto A$, we can see that the new state vectors which describe the learner's environment would not be correctly used, nor would the new actions be correctly evaluated by $\pi_{(3vs2,final)}$. In order to use the learned policy we modify it to handle the new actions and new state values in the second task so that the CMAC can reasonably evaluate them.

The CMAC function approximator takes a state vector and an action and returns the expected total reward. The learner can evaluate each potential action for the current S and then use π to choose one. We modify the weights in the tile coding so that when we input a 4 vs. 3 action the weights for the activated tiles are not zero but instead are initialized by $\pi_{3vs2,final}$. To accomplish this, we copy weights from the tiles which would be activated for a similar action in 3 vs. 2 into the tiles activated for every new action. The weights corresponding to the tiles that are activated for the "pass to teammate 2" action are copied into the weights for the tiles that are activated to evaluate the "pass to teammate 3" action. The modified CMAC will initially be unable to distinguish between these two actions.

To handle new state variables we follow a similar strategy. The 13 state variables which are present in 3 vs. 2 are already handled by the CMAC's weights. The weights for tiles activated by the six new 4 vs. 3 state variables are initialized to values of weights activated by similar 3 vs. 2 state variables. For instance, weights which correspond to "distance to teammate 2" values in the state representation are copied into the weights for tiles that are used to evaluate "distance to teammate 3" state values. This is done for all six new state variables. In this way, the tiles which correspond to every value in the new 4 vs. 3 state vector have been initialized to values determined via training in 3 vs 2 and can therefore be considered in the computation. See Table 1 for examples of mappings used. Identifying similar actions and states between two tasks is essential for constructing ρ and may prove to be the main limitation when attempting to apply behavior transfer to different domains.

Having constructed a ρ which handles the new states and actions, we can now set $\rho(\pi_{(3vs2,final)}) = \pi_{(4vs3,initial)}$. We do not claim that these initial CMAC weights are correct (and empirically they are not), but instead that the constructed CMAC allows the learner to more quickly discover a near-optimal policy.

Results and Discussion

To test the effect of loading the 3 vs. 2 CMAC weights into 4 vs. 3 keepers, we run a number of 3 vs. 2 episodes, save the CMAC weights ($\pi_{(3vs2,final)}$) from a random 3 vs. 2 keeper, and load the CMAC weights into all four keepers[2] in 4 vs. 3 so that $\rho(\pi_{(3vs2,final)}) = \pi_{(4vs3,initial)}$. Then we train on the 4 vs. 3 keepaway task until the average hold time for 1,000 episodes is greater than 9.0 seconds. To overcome the high variance inherent in the environment and therefore the noise in our evaluation, we run at least 100 independent trials for each number of 3 vs. 2 training episodes.

Table 2 reports the average time spent training 4 vs. 3 to achieve a 9.0 second average hold time for different amounts of 3 vs. 2 training. The middle column reports the time spent training on the 4 vs. 3 task while the third column shows the total time taken to train 3 vs. 2 and 4 vs. 3. As can be seen from the table, spending time training in the simpler 3 vs. 2 domain can cause the time spent in 4 vs. 3 to decrease. This shows that $time(S_2, s_{(2,initial)}, A_2, T, R, \rho(\pi_{(1,final)})) < time(S_2, s_{(2,initial)}, A_2, T, R, \pi_0)$.

# of 3 vs. 2 episodes	Ave. 4 vs. 3 time (hours)	Ave. total time (hours)
0	16.44	16.44
100	13.34	13.53
250	12.45	12.95
500	11.44	12.50
1,000	9.81	11.95
2,000	7.57	12.10
3,000	5.63	12.77
9,000	15.07	43.46

Table 2: Results from learning keepaway with different amounts of 3 vs. 2 training time indicate that behavior transfer can reduce training time.

Table 2 shows the potential of behavior transfer. We use a t-test to determine that the differences in the distributions of 4 vs. 3 training times and total training times when using behavior transfer are statistically significant ($p < 6 * 10^{-7}$) when compared to training 4 vs. 3 from scratch. Not only is the time to train the 4 vs. 3 task decreased when we first train on 3 vs. 2, but the total training time is less than the time to train 4 vs. 3 from scratch. We can therefore conclude that in the keepaway domain training first on a simpler task can increase the rate of learning enough that the total training time is decreased.

We would like to be able to determine the optimal amount of time needed to train on an easier task to speed up a more difficult task. It is apparent that there is some number of 3 vs. 2 episodes which would minimize $time(S_2, s_{(2,initial)}, A_2, T, R, \rho(\pi_{(1,final)}))$. This value may be distinct from the value which would minimize $time(S_1, s_{(1,initial)}, A_1, T, R, \pi_0) + time(S_2, s_{(2,initial)}, A_2, T, R, \rho(\pi_{(1,final)}))$. While it is not critical when considering the 4 vs. 3 task because many choices produce near optimal results, finding these values becomes increasingly difficult as well as increasingly

[2] We do so under the hypothesis that the policy of a single keeper represents all of the keepers' learned knowledge. Though in theory the keepers could be learning different policies that interact well with one another, so far there is no evidence that they do. One pressure against such specialization is that the keepers' start positions are randomized. In earlier informal experiments, there appeared to be some specialization when each keeper started in the same location every episode.

4 vs. 3 state variable	related 3 vs. 2 state variable
$dist(K_3, C)$	$dist(K_3, C)$
$dist(\mathbf{K_4}, C)$	$dist(K_3, C)$
Min($dist(K_3, T_1)$, $dist(K_3, T_2)$, $dist(K_3, \mathbf{T_3})$)	Min($dist(K_3, T_1)$, $dist(K_3, T_2)$)
Min($dist(\mathbf{K_4}, T_1)$, $dist(\mathbf{K_4}, T_2)$, $dist(\mathbf{K_4}, \mathbf{T_3})$)	Min($dist(K_3, T_1)$, $dist(K_3, T_2)$)

Table 1: This table describes part of the ρ transform from states in 3 vs. 2 keepaway to states in 4 vs. 3 keepaway. We denote the distance between a and b as $dist(a, b)$. Relevant points are the center of the field C, keepers K_1-K_4, and takers T_1-T_3. Keepers and takers are ordered in increasing distance from the ball and state values not present in 3 vs. 2 are in bold.

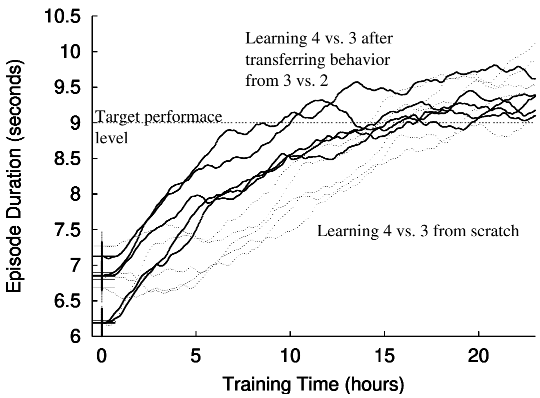

Figure 3: The learning curves for five representative keepers in the 4 vs. 3 keepaway domain when learning from scratch (dotted lines) have similar initial hold times when compared to five representative learning curves generated by transferring behavior from the 3 vs. 2 task (solid lines). The learners which have benefited from behavior transfer are able to more quickly learn the 4 vs. 3 task.

important when we scale up to larger tasks, such as 5 vs. 4 keepaway. Determining these training thresholds for tasks in different domains is currently an open problem and will be the subject of future research.

Interestingly, when the CMACs' weights are loaded into the keepers in 4 vs. 3, the initial hold times of the keepers do not differ much from the keepers with uninitialized CMACs, as shown in Figure 3. However, the information contained in the CMACs' weights prime the 4 vs. 3 keepers to more quickly learn their task. As the figure suggests, the 4 vs. 3 keepers which have loaded weights from 3 vs. 2 players learn at a faster rate than those 4 vs. 3 players that are training from scratch. This outcome suggests that the learned behavior is able to speed up the rate of reinforcement learning on the novel domain even though the knowledge we transfer is of limited initial value.

It is interesting that the required 4 vs. 3 training time for 9,000 episodes of 3 vs. 2 is greater than that of 1,000 episodes of 3 vs. 2. We posit this is due to overtraining; 4 vs. 3 must spend time "unlearning" some of the 3 vs. 2 specific knowledge before 4 vs. 3 can reach the hold time threshold. It makes intuitive sense that the 3 vs. 2 training would first learn policies that incorporate basic behaviors. We hypothesize that these simpler behaviors transfer well over to 4 vs. 3, but that more intricate behaviors learned after longer training periods are not as useful in 4 vs. 3 because they are more task dependent.

To test the sensitivity of the ρ function, we tried modifying it so that instead of copying the weights for the state variables for K_3 into the new 4 vs. 3 K_4 (see Table 1), we instead copy the K_2 state variable to this location. Now $\pi_{4vs3,initial}$ will evaluate the state variables for the closest and furthest keeper teammates to the same value instead of the two furthest teammates. Similarly, instead of copying weights corresponding to T_2 into the T_3 location, we copy weights from T_1. Training on 1,000 3 vs. 2 episodes and using $\rho_{modified}$ to train in 4 vs. 3, the total training time increased to 13.82 hours. Although this $\rho_{modified}$ outperforms training from scratch (with a statistical significance of $p < 0.004$), the total training time is 10%-20% longer compared to using ρ. Choosing non-optimal mappings between actions and states when constructing ρ seems to have a detrimental, but not necessarily disastrous, effect on the training time.

Initial results in scaling to 5 vs. 4 keepaway show that behavior transfer will work for this task as well. The average time for 5 vs. 4 keepaway to reach a hold time of 7.5 seconds on the same 25m x 25m field is 18.01 hours. However, if we train 4 vs. 3 from scratch for 1,000 episodes and set $\rho(\pi_{4vs3,final}) = \pi_{5vs4,initial}$, the average training time for 5 vs. 4 is reduced to 13.12 hours and the total training time is reduced to 14.98 hours. The difference in the total training times is statistically significant ($p < 2*10^{-5}$). We anticipate that behavior transfer will further reduce the total training time necessary to learn 5 vs. 4 as we tune the number 4 vs. 3 episodes as well as incorporate 3 vs. 2 training.

Related Work

The concept of seeding a learned behavior with some initial simple behavior is not new. There have been approaches to simplifying reinforcement learning by manipulating the transition function, the agent's initial state, and/or the reward function. Directed training (Selfridge, Sutton, & Barto 1985) is a technique to speed up learning whereby a human is allowed to change the task by modifying the transition function T. Using this method a human supervisor can gradually increase the difficulty of a task while using the same policy as the initial control for the learner. For instance, balancing a pole may be made harder for the learner by decreasing the mass or length of the pole. The learner will adapt to the new task faster using a policy trained on a related task than if learning from scratch.

Learning from easy missions (Asada *et al.* 1994) allows a human to change the start state of the learner, $s_{initial}$, making the task incrementally harder. Starting the learner near the exit of a maze and gradually allowing the learner to start further and further from the goal is an example of this. This kind of direction allows the learner to spend less total time learning to perform the final task.

Another successful idea, reward shaping (Colombetti & Dorigo 1993; Mataric 1994), also contrasts with behavior transfer. In shaping, learners are given an artificial problem which will allow the learner to train faster than on the actual problem which has different environmental rewards, R. Behavior transfer differs in intent in that we aim to transfer behaviors from existing, relevant tasks which can have different state and action spaces rather than creating artificial problems which are easier for the agent to learn. Furthermore, behavior transfer does not preclude the modification of the transition function, the start state, or the reward function and can therefore be combined with the other methods if desired.

Learned subroutines have been successfully transfered in a hierarchical reinforcement learning framework (Andre & Russell 2002). By analyzing two tasks, subroutines may be identified which can be directly reused in a second task that has a slightly modified state space. The learning rate for the second task can be substantially increased by duplicating the local sub-policy. This work can be thought of as another example for which ρ has been successfully constructed, but in a very different way.

Another related approach (Guestrin *et al.* 2003) uses linear programming to determine value functions for classes of similar agents. Using the assumption that T and R are similar among all agents of a class, class-based value subfunctions are inserted into agents in a new world which has a different number of objects (and thus different state and action spaces). Although no learning is performed in the new world, the previously learned value functions may still perform better than a baseline handcoded strategy. However, as the authors themselves state, the technique will not perform well in heterogeneous environments or domains with "strong and constant interactions between many objects (e.g. Robocup)." Our work is further differentiated as we continue learning in the second domain after performing ρ. While the initial performance in the new domain may be increased after loading learned value functions compared to learning from scratch, we have found that a main benefit is an increased learning rate.

Conclusions

We have introduced the behavior transfer method of speeding up reinforcement learning and given empirical evidence for its usefulness. We have trained learners using reinforcement learning in related tasks with different state and action spaces and shown that not only is the time to learn the final task reduced, but that the total training time is reduced using behavior transfer when compared to learning the final task from scratch.

Acknowledgments

We would like to thank Gregory Kuhlmann for his help with the experiments described in this paper as well as Nick Jong, Raymond Mooney, and David Pardoe for helpful comments and suggestions. This research was supported in part by NSF CAREER award IIS-0237699.

References

Albus, J. S. 1981. *Brains, Behavior, and Robotics*. Peterborough, NH: Byte Books.

Andre, D., and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 119–125.

Andre, D., and Teller, A. 1999. Evolving team Darwin United. In Asada, M., and Kitano, H., eds., *RoboCup-98: Robot Soccer World Cup II*. Berlin: Springer Verlag.

Asada, M.; Noda, S.; Tawaratsumida, S.; and Hosoda, K. 1994. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *Proc. of IAPR/IEEE Workshop on Visual Behaviors-1994*, 112–118.

Colombetti, M., and Dorigo, M. 1993. Robot Shaping: Developing Situated Agents through Learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley, CA.

Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003. Generalizing plans to new environments in relational mdps. In *The International Joint Conference on Artificial Intelligence (IJCAI)*.

Kuhlmann, G., and Stone, P. 2004. Progress in learning 3 vs. 2 keepaway. In Polani, D.; Browning, B.; Bonarini, A.; and Yoshida, K., eds., *RoboCup-2003: Robot Soccer World Cup VII*. Berlin: Springer Verlag.

Mataric, M. J. 1994. Reward functions for accelerated learning. In *International Conference on Machine Learning*, 181–189.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Riedmiller, M.; Merke, A.; Meier, D.; Hoffman, A.; Sinner, A.; Thate, O.; and Ehrmann, R. 2001. Karlsruhe brainstormers—a reinforcement learning approach to robotic soccer. In Stone, P.; Balch, T.; and Kraetszchmar, G., eds., *RoboCup-2000: Robot Soccer World Cup IV*. Berlin: Springer Verlag.

Selfridge, O.; Sutton, R. S.; and Barto, A. G. 1985. Training and tracking in robotics. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* 670–672.

Stone, P., and Sutton, R. S. 2002. Keepaway soccer: a machine learning testbed. In Birk, A.; Coradeschi, S.; and Tadokoro, S., eds., *RoboCup-2001: Robot Soccer World Cup V*. Berlin: Springer Verlag.

Stone, P.; Riley, P.; and Veloso, M. 2000. The CMUnited-99 champion simulator team. In Veloso, M.; Pagello, E.; and Kitano, H., eds., *RoboCup-99: Robot Soccer World Cup III*. Berlin: Springer. 35–48.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. MIT Press.