Transferring Instances for Model-Based Reinforcement Learning

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

ABSTRACT

Reinforcement learning agents typically require a significant amount of data before performing well on complex tasks. *Transfer learning* methods have made progress reducing sample complexity, but they have only been applied to model-free learning methods, not more data-efficient model-based learning methods. This paper introduces TIMBREL, a novel method capable of transferring information effectively into a model-based reinforcement learning algorithm. We demonstrate that TIMBREL can significantly improve the sample complexity and asymptotic performance of a model-based algorithm when learning in a continuous state space.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

Transfer Learning, Model Transfer, Reinforcement Learning

1. INTRODUCTION

In many situations, an agent must learn to execute a series of sequential actions, which is typically framed as a *reinforcement learning* [18] (RL) problem. Although RL approaches have enjoyed past successes (e.g., TDGammon [22], inverted Helicopter control [9], and robot locomotion [7]), they frequently take substantial amounts of data to learn a reasonable control policy. In many domains, collecting such data may be slow, expensive, or infeasible, motivating the need for sample-efficient learning methods.

One recent approach to speeding up RL so that it can be applied to difficult problems with large, continuous state spaces is *transfer learning* (TL). TL is a machine learning paradigm that reuses knowledge gathered in a previous source task to better learn a novel, but related, target task. Recent empirical successes in a variety of RL domains [12, 20, 23] have shown that transfer can significantly increase an agent's ability to learn quickly, even if agents in the two tasks have different available sensors or actions. However, to the best of our knowledge, TL methods have thus far been applied only to model-free RL algorithms.

Cite as: Transferring Instances for Model-Based Reinforcement Learning, Matthew E. Taylor, Nicholas K. Jong, and Peter Stone, *Proceedings of the ALAMAS+ALAG 2008 workshop at AAMAS 2008*, May, 12-16., 2008, Estoril, Portugal.

Model-free algorithms such as Q-Learning [24] and Sarsa [13, 15] learn to predict the utility of each action in different situations, but they do not learn the effects of actions. In contrast, model-based (or model-learning) methods, such as PEGASUS [10], R-MAX [3], and Fitted R-MAX [5], use their experience to learn an internal model of how the actions affect the agent and its environment, an approach empirically shown to often be more sample efficient. Such a model can be used in conjunction with *dynamic programming* [2] to perform off-line planning, often enabling superior action selection without requiring additional environmental samples. Building these models may be computationally intensive, but using CPU cycles to reduce data collection time is a highly favorable tradeoff in many domains (such as physically embodied agents). In order to further reduce sample complexity and ultimately allow RL to be applicable in more complex domains, this paper introduces *Transferring Instances for Model-Based REinforcement Learning* (TIMBREL), a novel approach to combining TL with model-based RL.

The key insight behind TIMBREL is that data may be reused between different tasks. Data is first recorded in a source task, transformed so that it applies to a target task, and then used by the target task learner as it builds its model. In this paper we utilize Fitted R-MAX, an instance based model-learning algorithm, and show how TIMBREL can help construct a target task model by using source task data. TIMBREL combines the benefits of transfer with those of model-based learning to reduce sample complexity. It works in continuous state spaces and is applicable when the source and target tasks have different state variables and action spaces. We fully implement and test our method in a set of Mountain Car tasks, demonstrating that transfer can significantly reduce the sample complexity of learning.

The rest of this paper is organized as follows. Section 2 provides a brief background of RL and Fitted R-MAX, as well as discussing a selection of related TL methods. The experimental domain is detailed in Section 3. Section 4 introduces TIMBREL and discusses its implementation when using Fitted R-MAX. Experimental results are presented in Section 5. Section 6 discusses possible future directions and concludes.

2. BACKGROUND AND RELATED WORK

In this paper we use the notation of *Markov decision processes* (MDPs). At every time step the agent observes its state $s \in S$ as a vector of k *state variables* such that $s = \langle x_1, x_2, \ldots, x_k \rangle$. In episodic tasks there is a starting state $s_{initial}$ and often a goal state s_{goal}, which terminates the episode if reached by the agent. The agent selects an action from the set of available actions A at every time step. The start and goal states may be generalized to sets of states. A task also defines the reward function $R : S \times A \mapsto \mathbb{R}$, and

the transition function $T : S \times A \mapsto S$ fully describes the dynamics of the system. The agent will attempt to maximize the long-term reward determined by the (initially unknown) reward function R and the (initially unknown) transition function T.

A learner chooses which action to take in a state via a policy, $\pi : S \mapsto A$. π is modified by the learner over time to improve performance, which is defined as the expected total reward. Instead of learning π directly, many RL algorithms instead approximate the action-value function, $Q : S \times A \mapsto \mathbb{R}$, which maps state-action pairs to the expected real-valued return. If the agent has learned the optimal action-value function, it can select the optimal action from any state by executing the action with the highest action-value.

In this paper, we introduce and utilize TIMBREL to improve the performance of Fitted R-MAX [5], an algorithm that approximates the action-value function Q for large or infinite state spaces by constructing an MDP over a small (finite) sample of states $X \subset S$. For each sample state $\mathsf{x} \in X$ and action $a \in A$, Fitted R-MAX estimates the dynamics $T(\mathsf{x}, a)$ using all the available data for action a and for states s near x.[1] Some generalization from nearby states is necessary because we cannot expect the agent to be able to visit x enough times to try every action. As a result of this generalization process, Fitted R-MAX first approximates $T(\mathsf{x}, a)$ as a probability distribution over predicted successor states in S. A value approximation step then approximates this distribution of states in S with a distribution of states in X. The result is a stochastic MDP over a finite state space X, with transition and reward functions derived from data in S. Applying dynamic programming to this MDP yields an action-value function over $X \times A$ that can be used to approximate the desired action-value function Q. For the original 2D Mountain Car task, Fitted R-MAX learns policies using less data than many existing model-free algorithms [5].

Approaches that transfer between model-free RL algorithms are most closely related to TIMBREL. Torrey et al. [23] show how to automatically extract *advice* from a source task by identifying actions which have higher Q-values than other available actions; this advice is then mapped by a human to the target task as initial preferences given to the target task learner. In our previous work [20], we learn an action-value function for a source task, translate the function into a target task via a hand-coded *inter-task mapping*, and then use the transferred function to initialize the target task agent. Other work [12] shows that in *relational reinforcement learning*, object-specific action-value functions can be used for initialization when the number of objects change between the source and target tasks. In all three cases the transferred knowledge is effectively used to improve learning in the target task, but only using model-free learning methods that inherently require more data than model-based learning.

3. GENERALIZED MOUNTAIN CAR

This section introduces our experimental domain, a generalized version of the standard RL benchmark Mountain Car task [15]. Mountain Car is an appropriate testbed for TIMBREL with Fitted R-MAX because it is among the simplest continuous domains that can benefit from model-based learning, and it is easily generalizable to enable TL experiments.

In Mountain Car, the agent must generalize across continuous state variables in order to drive an underpowered car up a Mountain to a goal state. We also introduce 3D Mountain Car as extension

[1]Fitted R-MAX is an instance-based learning method; our implementation currently retains all observed data to compute the model. In the future we plan to enhance the algorithm so that instances can be discarded without significantly decreasing model accuracy.

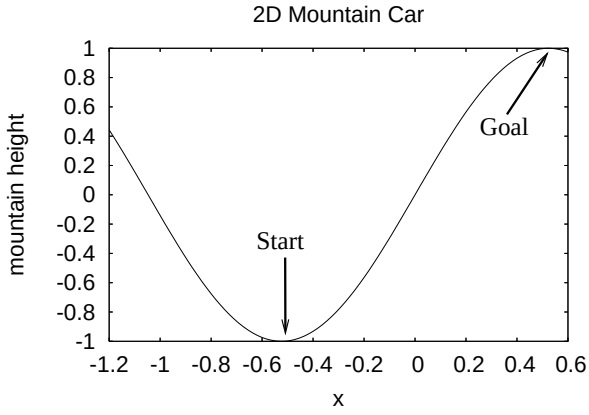

Figure 1: In the standard 2D Mountain Car the agent must travel along a curve (Mountain).

of the 2D task, retaining much of the structure of the 2D problem. In both tasks the transition and reward functions are initially unknown. The agent begins at rest at the bottom of the hill.[2] The reward for each time step is -1. The episode ends, and the agent is reset to the start state, after 500 time steps or if reaches the goal state.

3.1 Two Dimensional Mountain Car

In the two dimensional Mountain Car task, two continuous variables fully describe the agent's state. The horizontal position (x) and velocity ($\dot{x}$) are restricted to the ranges $[-1.2, 0.6]$ and $[-0.07, 0.07]$ respectively. The agent may select one of three actions on every timestep; {`Left`, `Neutral`, `Right`} change the velocity by -0.001, 0, and 0.001 respectively. Additionally, $-0.025(\cos(3x))$ is added to $\dot{x}$ on every timestep to account for the force of gravity on the car. The start state is ($x = -\pi/6, \dot{x} = 0$), and the goal states are those where $x \geq 0.5$ (see Figure 1). We use the publicly available[3] version of this code for our experiments.

3.2 Three Dimensional Mountain Car

To create a three dimensional task, we extend the Mountain's curve into a surface (see Figure 2). The state is composed of four continuous state variables: $x, \dot{x}, y, \dot{y}$. The positions and velocities have ranges of $[-1.2, 0.6]$ and $[-0.07, 0.07]$, respectively. The agent selects from five actions at each timestep: {`Neutral`, `West`, `East`, `South`, `North`}. West and East modify $\dot{x}$ by -0.001 and +0.001 respectively, while South and North modify $\dot{y}$ by -0.001 and +0.001 respectively.[4] The force of gravity adds $-0.025(\cos(3x))$ and $-0.025(\cos(3y))$ on each time step to $\dot{x}$ and $\dot{y}$, respectively. The goal state region is defined by $x \geq 0.5$ and $y \geq 0.5$.

This task is more difficult than the 2D task because of the increased state space size and additional actions. Furthermore, since the agent can affect its acceleration in only one of the two spacial dimensions at any given time, one cannot simply "factor" this problem into the simpler 2D task. While data gathered from the 2D task

[2]Both Mountain Car tasks are deterministic, as is Fitted R-MAX. To introduce randomness and allow multiple learning trials, when each domain is initialized, x (and y in 3D) in the start state is perturbed by a random number in $[-0.005, 0.005]$.

[3]Available at `http://rlai.cs.ualberta.ca/RLR/MountainCarBestSeller.html`

[4]Although we call the agent's vehicle a "car," it does not turn but simply accelerates in the four cardinal directions.

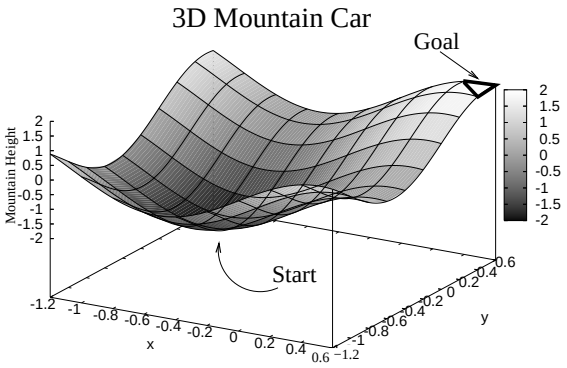

Figure 2: In 3D Mountain Car the 2D curve becomes a 3D surface. The agent starts at the bottom of the hill with no kinetic energy and attempts to reach the goal area in the Northeast corner.

should be able to help an agent learn the 3D task, we do expect that some amount of learning will be required after transfer.

3.3 Learning Mountain Car

Our experiments used Fitted R-MAX to learn policies in the Mountain Car tasks. We began by replicating the methods and result of applying Fitted R-MAX to 2D Mountain Car task as reported in the literature [5]. To apply Fitted R-MAX to 3D Mountain Car, we first scaled the state space so that each dimension ranges over the unit interval, effectively scaling the state space to a unit hypercube. We sampled a finite state space from this hypercube by applying a grid where each position state variable can be one of 8 values, and each velocity state variable can be one of 9 values. The 3D version of Mountain Car has 2 of each type of state variable; we obtained a sample X of $8^2 \times 9^2 = 5184$ states that approximated the original state space state S. For any state $\mathsf{x} \in X$ and action $a \in A$, Fitted R-MAX estimates $T(\mathsf{x}, a)$ using a probability distribution over observerd (s_i, a, r_i, s'_i) instances in the data available for action a. Each instance i is given a weight w_i depending on the Euclidean distance from x to s_i and on the *model breadth* parameter b, according to the following formula:

$$w_i \propto e^{-\left(\frac{|\mathsf{x}-s_i|}{b}\right)^2}.$$

Intuitively, b controls the degree of generalization used to estimate $T(\mathsf{x}, a)$ from nearby data. In 3D Mountain Car experiments, we used a parameter of $b = 0.1$. In theory, all instances that share the action a could be used to help approximate x, where each instance i's contribution is modified by w_i (i.e., a Gaussian weighting that exponentially penalizes distance from x). To reduce the computational cost of the algorithm, for a given state x we computed the weights for the nearest instances first. Once an instance's weight failed to increase the cumulative weight by at least 40%, we ignored the remaining instances' contribution as negligible. Finally, when the accumulated weight failed to reach a threshold of 1.0, we used Fitted R-MAX's exploration strategy of assuming an optimistic transition to a maximum-reward absorbing state.

Changing the learning parameters for Fitted R-MAX outlined above affect three primary aspects of learning:

- How accurately the optimal policy can be approximated.
- How many samples are needed to accurately approximate the best policy, given the representation.
- How much computation is required when performing dynamic programming.

For this work, it was most important to find settings which allowed the agent to learn a reasonably good policy in relatively few episodes so that we could demonstrate the effectiveness of TIMBREL on sample complexity. We do not argue that the above parameters are optimal. They could be tuned to emphasize any of the above goals, such as achieving higher performance in the limit. In preliminary results (not shown), we compared using Fitted R-MAX and to using model-free ϵ-greedy Sarsa(λ). [5] Fitted R-MAX learned to consistently find the goal state with roughly two orders of magnitude less data than Sarsa, although learning with Fitted R-MAX takes substantially more computational resources than Sarsa.

4. MODEL TRANSFER

Model-based algorithms learn to estimate the transition model of an MDP, predicting the effects of actions. The goal of transfer for model-based RL algorithms is to allow the agent to build such a model from data gathered both in a previous task, as well as in the current task. To help frame the exposition, we note that transfer methods must typically perform the following three steps:

I Use the source task agent to record some information during, after, or about, learning. Successful TL approaches include recording learned action-value functions or higher-level advice about high-value policies.

II Transform the saved source task information so that it applies to the target task. This step is most often necessary if the states and actions in the two tasks are different, as considered in this paper.

III Utilize the transformed information in the target task. Successful approaches include using source task information to initialize the learner's action-value function, giving advice about actions, and suggesting potentially useful sequences of actions (i.e., *options*).

Section 4.1 introduces TIMBREL, a novel transfer method, which accomplishes these steps. Section 4.2 gives an overview of the method details how TIMBREL is used in the Mountain Car domain with Fitted R-MAX, our chosen model-based RL algorithm.

4.1 Instance-Based Model Transfer

This section provides an overview of TIMBREL. In order to transfer a model, our method takes the novel approach of transferring observed instances from the source task. The tuples, in the form (s, a, r, s'), describe experience the source task agent gathered while interacting with its environment (Step I). One advantage of this approach as compared to transferring an action-value function or a full environmental model (e.g., the transition function) is that the source task agent is not tied to a particular learning algorithm or representation: whatever RL algorithm that learns will necessarily have to interact with the task and collect experience. This flexibility allows a source task algorithm to be selected based on characteristics of the task, rather than on demands of the transfer algorithm.

To translate a source task tuple into an appropriate target task tuple (Step II) we utilize *inter-task mappings* [20], which have been

[5] Specifically, we used a CMAC [1] function approximator with 14 4-dimensional linear tilings, which is analogous to how Singh and Sutton [15] used 14 2-d dimensional linear tile codings for their 2D task.

Algorithm 1 TIMBREL Overview

```
1: Learn in the source task, recording (s, a, r, s') transitions.
2: Provide recorded transitions to the target task agent.
3: while training in the target task do
4:   if the model-based RL algorithm is unable to accurately estimate some T(x, a) or R(x, a) then
5:     while T(x, a) or R(x, a) does not have sufficient data do
6:       Locate one or more saved instances that, according to the inter-task mappings, are near the current x, a to be estimated.
7:       if no such unused source task instances exist then
8:         exit the while starting on line 5
9:       Use x, a, the saved source task instance, and the mappings to translate the saved instance into one appropriate to the target task.
10:      Add the transformed instance to the current model for x, a.
```

successfully used in past transfer learning research to specify how pairs of tasks are related via an action mapping and a state variable mapping. This pair of mappings identifies source task actions which have similar effects as target task actions, and allows a mapping of target task state variables into source task state variables.

When learning in the target task, TIMBREL specifies when to use source task instances to help construct a model of the target task (Step III). Briefly, when insufficient target task data exists to estimate the effect of a particular (x, a) pair, instances from the source task are transformed via an action-dependant inter-task mapping, and are then treated as a previously observed transition in the target task model. The TIMBREL method is summarized in Algorithm 1.

Notice that TIMBREL performs the translation of data from the source task to the target task (line 10) on-line while learning the target task. While the translation step of transfer algorithms is more commonly performed off-line before training in the target task, this just-in-time approach is appropriate because of how the mappings are utilized. In the following section, we detail how the current state x that is being approximated will affect how the source task sample is translated. By only transferring instances that will be immediately used in the target task, the amount of computation needed is limited. Furthermore, this method will minimize the number of source instances that must be reasoned over in the target task model by only transferring necessary source task data.

4.2 TIMBREL Implementation

In this section we detail how TIMBREL is used to transfer between tasks in the Mountain Car domain when using Fitted R-MAX as the underlying RL algorithm. Although TIMBREL is a domain-independent transfer method which is designed to be compatible with multiple model-learning RL algorithms, we will ground our exposition in the context of Fitted R-MAX and Mountain Car. Throughout this section we use the subscript $\mathbb{S}$ to denote actions, states, and state variables in the source task, and the subscript $\mathbb{T}$ for the target task.

The core result of this paper is to demonstrate transfer between the standard 2D Mountain Car task and the 3D Mountain Car task. After learning the 2D task, TIMBREL must be provided an inter-task mapping between the two tasks. The action mapping, χ_A, maps a target task action into a source task action: $\chi_A(a_\mathbb{T}) = a_\mathbb{S}$, and χ_S maps a target task state variable into a source task state variable: $\chi_S(s_{(i,\mathbb{T})}) = s_{(j,\mathbb{S})}$. In this work we assume that the inter-task mapping in Table 1 is provided to the agent, but other work [19] has demonstrated that the same mapping may be learned

Inter-task Mapping for Mountain Car

Action Mapping	State Variable Mapping
χ_A(Neutral) = Neutral	$\chi_S(x) = x$
χ_A(North) = Right	$\chi_S(\dot{x}) = \dot{x}$
χ_A(East) = Right	or
χ_A(South) = Left	$\chi_S(y) = x$
χ_A(West) = Left	$\chi_S(\dot{y}) = \dot{x}$

Table 1: This table describes the mapping used by TIMBREL to construct target task instances from source task data.

autonomously in this domain with relatively little overhead. Note that the state variable mapping is defined so that either the target task state variables (x and $\dot{x}$) *or* (y and $\dot{y}$) are mapped into the source task. As we will discuss, the unmapped target task state variables will be set by the state variables' values in the state x that we wish to approximate.

As discussed in Section 2, Fitted R-MAX approximates transitions from a set of sample states $\mathsf{x} \in X$ for all actions. When the agent initially encounters the target task, no target task instances are available to approximate T. Without transfer, Fitted R-MAX would be unable to approximate $T(\mathsf{x}_\mathbb{T}, a_\mathbb{T})$ for any x and would set the value of $Q(s_\mathbb{T}, a_\mathbb{T})$ to an optimistic value (R_{max}) to encourage exploration. Instead, TIMBREL is used to generate target instances to help approximate $T(\mathsf{x}_\mathbb{T}, a_\mathbb{T})$.

TIMBREL provides a set of source task instances, as well as the inter-task mappings, and must construct one or more target task tuples, $(s_\mathbb{T}, a_\mathbb{T}, r, s'_\mathbb{T})$, to help approximate $T(\mathsf{x}_\mathbb{T}, a_\mathbb{T})$. The goal of transfer is to find some source task tuple $(s_\mathbb{S}, a_\mathbb{S}, r, s'_\mathbb{S})$ where $a_\mathbb{S} = \chi_A(a_\mathbb{T})$ and $s_\mathbb{S}$ is "near" $s_\mathbb{T}$ (line 6). Once we identify such a source task tuple, we can then use χ^{-1} to convert the tuple into a transition appropriate for the target task (line 10), and use it to help approximate T (line 11).

As an illustrative example, consider the case when the agent wants to approximate $T(\mathsf{x}_\mathbb{T}, a_\mathbb{T})$, where $\mathsf{x}_\mathbb{T} = \langle x_\mathbb{T}, y_\mathbb{T}, \dot{x}_\mathbb{T}, \dot{y}_\mathbb{T} \rangle = \langle -0.6, -0.2, 0, 0.1 \rangle$ and $a_\mathbb{T}$ = East. TIMBREL considers source task transitions that contain the action Right. χ_S is defined so that either the x or y state variables can be mapped from the target task to the source task, which means that we should consider two transitions selected from the source task instances. The first tuple is selected to minimize the Euclidean distances $D(x_\mathbb{T}, x_\mathbb{S})$ and $D(\dot{x}_\mathbb{T}, \dot{x}_\mathbb{S})$, where each distance is scaled by the range of the state variable. The second tuple is chosen to minimize $D(y_\mathbb{T}, x_\mathbb{S})$ and $D(\dot{y}_\mathbb{T}, \dot{x}_\mathbb{S})$.

Continuing the example, suppose that the first source task tuple selected was

$$(\langle -0.61, 0.01 \rangle, \text{Right}, -1, \langle -0.59, 0.02 \rangle).$$

If the inter-task mapping were defined so that both the x and y state variables simultaneously, the inverse inter-task mapping *could* be used to convert the tuple into

$$(\langle -0.61, -0.61, 0.01, 0.01 \rangle, \text{East}, -1, \langle -0.59, -0.59, 0.02, 0.02 \rangle).$$

However, this point is not near the current $\mathsf{x}_\mathbb{T}$ we wish to approximate. Instead, we recognize that this sample was selected from the source task to be near to $\mathsf{x}_\mathbb{T}$ and $\dot{x}_\mathbb{T}$, and transform the tuple, assuming that $y_\mathbb{T}$ and $\dot{y}_\mathbb{T}$ are kept constant. With this assumption, we form the target task tuple

$$(\langle -0.61, y_\mathbb{T}, 0.01, \dot{y}_\mathbb{T} \rangle, \text{East}, -1, \langle -0.59, y_\mathbb{T}, 0.2, \dot{y}_\mathbb{T} \rangle) =$$
$$(\langle -0.61, -0.2, 0.01, 0 \rangle, \text{East}, -1, \langle -0.59, -0.2, 0.02, 0 \rangle).$$

The analogous step is then performed for the second selected source task tuple; we transform the source task tuple with χ while assuming that $x_{\mathbb{T}}$ and $\dot{x}_{\mathbb{T}}$ are held constant. Finally, both transferred instances are added to the approximation of $T(\mathsf{x}, a)$.

TIMBREL thus transfers pairs of source task instances to help approximate the transition function. Other model-learning methods may need constructed trajectories instead of individual instances, but TIMBREL is able to generate trajectories as well. Over time, the learner will approximate $T(\mathsf{x}_{\mathbb{T}}, a_{\mathbb{T}})$ for different values of (x, a) in order to construct a model for the target task environment. Any model produced via this transfer may be incorrect, depending on how representative the saved source task instances are of the target task (as modified by χ). However, our experiments demonstrate that using transferred data may allow a model learner to produce a model that is more accurate than if the source data were ignored.

As discussed in Section 3.3, Fitted R-MAX uses the distance between instances and x to calculate instance weights. When an instance is used to approximate x, that instance's weight is added to the total weight of the approximation. If the total weight for an approximation does not reach a threshold value of 1.0, an optimistic value (R_{max}) is used because not enough data exists for an accurate approximation. When using TIMBREL, the same calculation is performed, but now instances from both the source task and target task can be used.

As the agent interacts with the target task, more transitions are recorded and the approximations of the transition function at different (x, a) pairs need to be recalculated based on the new information. Each time an approximation needs to be recomputed, Fitted R-MAX first attempts to use only target task data. If the number of instances available (where instances are weighted by their distance from x) does not exceed the total weight threshold, source task data is transferred to allow an approximation of $T(\mathsf{x}_{\mathbb{T}}, a_{\mathbb{T}})$. This process is equivalent to removing transferred source task data from the model as more target task data is observed and therefore allows the model's accuracy to improve over time. Again, if the total weight from source task and target tasks instances for an approximated x does not reach 1.0, R_{max} is assigned to the model for x.

As a final implementation note, consider what happens when some x maps to an $s_{\mathbb{S}}$ that is not near any experienced source task data. If there are no source task transitions near $s_{\mathbb{S}}$, it is possible that using all available source task data will not produce an accurate approximation (recall that instance weights are proportional to the square of the distance from the instance to x). To avoid a significant reduction in performance with limited improvement in approximating T, we imposed a limit of 20 source task tuples when approximating a particular point (line 5). This threshold serves a similar purpose as the 10% cumulative weight threshold discussed in Section 3.3.

5. TRANSFER EXPERIMENTS

In order to test the efficacy of transfer, we conducted an experiment to measure the learning speed of Fitted R-MAX in the Mountain Car domain both with and without TIMBREL. To transfer from 2D Mountain Car into the more complex 3D Mountain Car, we first allow Fitted R-MAX to train for 100 episodes in the 2D task while recording all observed $\langle s, a, r, s' \rangle$ transitions. The agent's learning parameters were set so that the agent thoroughly explored the source task state space and only discovered the goal near the end of learning.[6]

[6]We experimented with 5 different parameter settings for Fitted R-MAX in the 2D Task. Recall that every episode lasts 500 time steps if the goal is not found. When learning 2D Mountain Car, the agent experienced 48,669 source task transitions during 100 episodes.

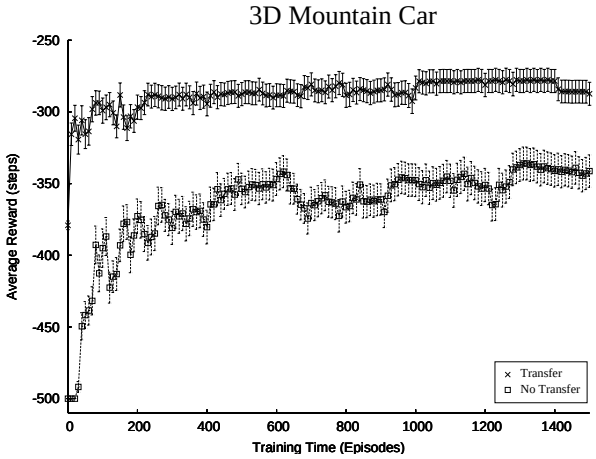

Figure 3: This figure shows that TIMBREL significantly improves the speed of Fitted R-MAX on the 3D Mountain Car. The average performance is plotted every 10 episodes, along with the standard error.

Roughly 100 preliminary experiments were run on the 3D task, each lasting a few hundred episodes, in order to select Fitted R-MAX settings for the non-transfer learner, which were discussed in Section 3.3. We ran 10 trials of Fitted R-MAX without transfer and 10 trials with transfer, each of for 1,500 episodes. After learning, we averaged each set of 10 independent learning curves, but due to the low number of trials, the learning curves were quite noisy. To improve the clarity of our results, we also smoothed the two summary learning curves by averaging over groups of 10 episodes. Figure 3 shows the summary of our two sets of experiments, along with the standard error at each point. We ran paired t-tests on the 151 graphed points and found that every difference was statistically significant ($p < 1.7 \times 10^{-4}$), which confirms that utilizing transfer between our pair of Mountain Car tasks yield a significant advantage for Fitted R-MAX.

Our algorithm and implementation have been designed to minimize the sample complexity. However, it is worth noting that there is a significant difference in the computational complexity of the transfer and non-transfer methods. Every time the transfer agent needs to use source task data to estimate T, it must locate the most relevant data and then insert it into the model. Additionally, the transfer agent has much more data available initially, and thus its dynamic programming step is significantly slower than the non-transfer agent. These factors cause the transfer learning trials to take roughly twice as much wall clock time as the non-transfer trials. While our code could be better optimized, using the additional transferred data will always slow down the agent, relative to an agent that is not using transfer, but is running for the same number of episodes. However, in many domains a tradeoff between computational and sample complexity is highly advantageous, and is one of the benefits inherent to model-based reinforcement learning.

Also note that the transfer and non-transfer learning curves do not end at the same performance. We do not claim that transfer has produced a superior asymptotic performance, however, because neither learning curve has fully converged. We expect that the non-transfer Fitted R-MAX agents would reach the same, or perhaps superior, performance. However, these results do demonstrate that transfer can provide a significant speed advantage.

6. CONCLUSION AND FUTURE WORK

In this paper we have introduced TIMBREL, which we believe to be the first transfer method compatible with model-based reinforcement learning. We demonstrate that when learning 3D Mountain Car with Fitted R-MAX, TIMBREL can significantly reduce the sample complexity and demonstrated how transfer is affected by changes to the source task's reward and transfer functions.

There are a number of research directions suggested by this work. When learning the 2D source task in this paper, we explicitly set the parameters to maximize exploration. It would be informative to study how transfer efficacy changes when the amount of exploration is decreased in the source task. This is an issue related to, but distinct from, discovering how the target task performance is affected when the number of source task episodes changes. A final question left for future work is whether one could determine if collecting additional samples in the source task would help learn the target, which could help reduce the total amount of data required to learn both tasks.

We predict that TIMBREL will work, possibly with minor modifications, in other model-based RL algorithms. In the future we would like to experiment with other model-based RL algorithms, such as R-MAX, to see if transfer is as effective as in Fitted R-MAX, and see if our methods need to be modified to accommodate the different model representation. Additionally, we intend to apply TIMBREL to more complex domains that have continuous state variables; we expect that transfer will provide even more benefit as task difficulty increases.

Acknowledgments

We would like to thank the anonymous reviewers for helpful comments and suggestions. This research was supported in part by DARPA grant HR0011-04-1-0035, NSF CAREER award IIS-0237699, and NSF award EIA-0303609.

7. REFERENCES

[1] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.

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

[3] R. I. Brafman and M. Tennenholtz. R-Max – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[4] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1017–1023, Cambridge, MA, 1996. MIT Press.

[5] N. K. Jong and P. Stone. Model-based exploration in continuous state spaces. In *The Seventh Symposium on Abstraction, Reformulation, and Approximation*, July 2007.

[6] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, pages 260–268. Morgan Kaufmann, San Francisco, CA, 1998.

[7] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

[8] Y. Liu and P. Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proc. of the 21st National Conf. on Artificial Intelligence*, July 2006.

[9] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[10] A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

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

[12] J. Ramon, K. Driessens, and T. Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proc. of The 18th European Conf. on Machine Learning*, 2007.

[13] G. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University, 1994.

[14] M. Saggar, T. D'Silva, N. Kohl, and P. Stone. Autonomous learning of stable quadruped locomotion. In G. Lakemeyer, E. Sklar, D. Sorenti, and T. Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434, pages 98–109. Springer Verlag, Berlin, 2007.

[15] S. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[16] V. Soni and S. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proc. of the Twenty First National Conf. on Artificial Intelligence*, July 2006.

[17] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

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

[19] M. E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2008.

[20] M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007.

[21] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007.

[22] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[23] L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 2005.

[24] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[25] W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.