In *The European Conference on Machine Learning & Principles and Pratice of Knowledge Discovery in Databases* (ECML PKDD 2008), Antwerp, Belgium, September 2008.

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
{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 primarily 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 efficiency and asymptotic performance of a model-based algorithm when learning in a continuous state space. Additionally, we conduct experiments to test the limits of TIMBREL's effectiveness.

1 Introduction

In many situations, an agent must learn to execute a series of sequential actions, which is typically framed as a *reinforcement learning* (RL) [1] problem. Although RL approaches have enjoyed past successes (e.g., TDGammon [2], inverted Helicopter control [3], and robot locomotion [4]), 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 [5–7] 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. Note that TL is related to, but distinct from, the *concept drift* [8] and *multi-task learning* [9, 10] paradigms. Concept drift assumes that the environment is non-stationary: at certain points in time, the environment may change arbitrarily and the change is unannounced. Multi-task learning assumes that the agent experiences many problems and that they are all drawn from the same distribution (and thus all tasks have the same actions and state variables). In contrast, TL methods generally assume that the agent is notified when the task changes and generally do not assume that source and target tasks are drawn from the same distribution.

Model-free algorithms such as Q-Learning [11] and Sarsa [12] 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 [13], R-MAX [14], and Fitted R-MAX [15], 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* [16] 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 gathered in a source task can be used to build beneficial models in a target task. 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. 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.

In principle, the core TIMBREL algorithm (Section 3.1) could be used with other model-learning algorithms, but we leave such extensions to future work. The experiments in this paper use TIMBREL by applying it to Fitted R-MAX (detailed in Section 5), as it can both learn in continuous state spaces and has had significant empirical success [15]. This paper's results thus demonstrate that TIMBREL works in continuous state spaces, as well as between tasks with different state variables and action spaces.

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. Section 3 introduces TIMBREL and the experimental domain is detailed in Section 4. Results are presented in Section 6. Section 7 discusses possible future directions and concludes.

2 Background and Related Work

In this paper we use the notation of *Markov decision processes* (MDPs) [17]. 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 [17]. 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 [15], 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 $\mathtt{x} \in X$ and action $a \in A$, Fitted R-MAX estimates the dynamics $T(\mathtt{x}, a)$ using all the available data for action a and for states s near $\mathtt{x}$.[1] Some generalization from nearby states is necessary because we cannot expect the agent to be able to visit $\mathtt{x}$ enough times to try every action. As a result of this generalization process, Fitted R-MAX first approximates $T(\mathtt{x}, a)$ 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. Past work [15] empirically shows that Fitted R-MAX learns policies using less data than many existing model-free algorithms.

Fitted R-MAX is summarized in Algorithm 1. s^{opt} is a dummy state that represents unexplored states (where $V(s^{\text{opt}})$ is set to R_{max}). s^{term} is a dummy absorbing state that all discovered terminal states get mapped to. D is a data structure that holds all observed instances. ϕ^{S^a} is an averaging object that approximates the effect of action a at state s using nearby sample transitions $d \in D^a$. ϕ^X is an averaging object that approximates the value of each predicted successor state using nearby sample states $x \in X$. The reader is referred to [15] for detailed descriptions of the update rules (lines 16 and 17).

Most similar to TIMBREL are methods that transfer between model-free RL algorithms with different state and action spaces. Torrey et. al [5] 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 past work [6], an agent learns an action-value function in a source task, translates the function into a target task via a hand-coded *inter-task mapping*, and then uses the transferred function to initialize the target task agent. Other recent

[1] Fitted R-MAX is an instance-based learning method; our implementation currently retains all observed data to compute the model. It could, in principle, be enhanced to automatically discard instances without significantly decreasing model accuracy.

Algorithm 1 Fitted R-MAX (R_{max}, r, b, *minFraction*, *explorationThreshold*)

$X \leftarrow \{s^{\text{opt}}, s^{\text{term}}\}$ # Initialize state sample
$X.InitializeUniformGrid(r)$
for all $a \in A$ **do** # Initialize experience sample
  $D^a \leftarrow \{\langle s^{\text{opt}}, a, V^{\max}, s^{\text{term}} \rangle\}$
loop
  $s \leftarrow$ initial state # Begin a trajectory
  $a \leftarrow \text{argmax}_a \left[R(s) + \sum_{x' \in X} P(x'|s,a)V(x') \right]$
  repeat
    Execute a
    Observe r and s'
    if s' is terminal **then**
      $s' \leftarrow s^{\text{term}}$
    else
      $a' \leftarrow \text{argmax}_a \left[R(s) + \sum_{x' \in X} P(x'|s,a)V(x') \right]$
    $D^a \leftarrow D^a \cup \{\langle s, a, r, s' \rangle\}$ # Update experience sample
    Update ϕ^X and ϕ^{S^a} via $\langle$experience, *minFraction*, and *explorationThreshold*$\rangle$
    Update estimates of R and P based on ϕ^X and ϕ^{S^a}
    Compute $V(x)$ for $x \in X$ via dynamic programming
    $s \leftarrow s'$
    $a \leftarrow a'$
  until s is a terminal state # the episode ends

work by Lazaric et. al [7] demonstrates that source task instances (that is, observed $\langle s, a, r, s' \rangle$ tuples) can be usefully transferred between tasks for a batch value-function learning algorithm. In all three cases the transferred knowledge is effectively used to improve learning in the target task, but only for model-free learning methods.

Atkeson and Santamaria [18] show that if only the reward function changed between tasks, a locally weighted regression model can be directly applied from a source task in a novel task. Tanaka and Yamamura [19] consider multi-task learning in a discrete state space. By recording the average and deviation of Q-values for all (s, a) pairs, agents in the $n + 1^{th}$ task can initialize their Q-values to the previously seen average to learn faster. Additionally, agents can order their prioritized sweeping [20] updates based on the average and deviation of each (s, a) pair to gain additional learning speed advantages.

Lastly, two works consider multi-task learning in a Bayesian model-based RL [21] setting. First, Sunmola and Wyatt [22] introduce two methods that use instances from previous tasks to set priors in a Bayesian learner. Initial experiments show that given an accurate estimation of the prior distributions, an agent may learn a novel task faster. Second, Wilson et. al [10] consider learning in a hierarchical Bayesian RL setting over multiple MDP distributions. By setting priors based on previously learned tasks, a new task in a particular distribution can be learned significantly faster. A simple parameterized reward function and the location of an absorbing goal state may change between the different tasks.

TIMBREL differs from previous work along a number of dimensions. Most important, inter-task mappings allow TIMBREL to transfer knowledge suitable for model-learning RL agents, even when transfer is between MDPs with different state variables and actions. Additionally, TIMBREL can run on-line, is not limited to discrete domains, and is designed for transfer (as opposed to multi-task learning).

3 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:

1. 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.
2. 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.
3. 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*).

The following section introduces TIMBREL, a novel transfer method, which accomplishes these steps. Later, in Section 5, we detail how TIMBREL is used in our test domain with Fitted R-MAX, our chosen model-based RL algorithm.

3.1 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 1). 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 2) we utilize *inter-task mappings* [6], which have been 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 3). Briefly, when insufficient target task data exists to estimate the effect of a particular $(\mathtt{x}, a)$ 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 2.

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. Transfer algorithms more commonly performed such translations off-line, before training in the target task, but this just-in-time approach is justified because of how the source data are utilized. In

Algorithm 2 TIMBREL Overview

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

Section 5, we detail how the current state, $\mathtt{x}$, will affect how the source task sample is translated in our particular task domain. Only transferring instances that will be immediately used in the target task helps to limit computational costs. 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 Generalized Mountain Car

This section introduces our experimental domain, a generalized version of the standard RL benchmark mountain car task [12]. 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 discuss 3D mountain car [23], an extension of the 2D task. 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 it reaches the goal state.

4.1 Two Dimensional Mountain Car

In the two dimensional mountain car task, two continuous variables fully describe the agent's state (see Figure 1). 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

[2] Both mountain car tasks are deterministic, and Fitted R-MAX's exploration uses a fixed random seed. 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 $[-0.005, 0.005]$.

by -0.0007, 0, and 0.0007 respectively.[3] Additionally, $-0.025(\cos(3x))$ is added to $\dot{x}$ on every timestep to account for the x-component of the force of gravity on the car, which depends on the local slope of the mountain. The start state is $(x = -\frac{\pi}{6}, \dot{x} = 0)$, and the goal states are those where $x \geq 0.5$. We use the publicly available[4] version of this code for our experiments.

The transfer experiments in this paper (Section 6) use three variants of the 2D mountain car task. The first, which we will call the *Standard 2D task* is described in the previous paragraph. The *No Goal 2D task* is the same as the standard task, except that goal state has been removed. This task will be used to show how the effectiveness of transfer changes when the reward function changes. The third variant, the *High Power 2D task*, changes the car so that the velocity is changed by ± 0.0015: the car has more than twice the acceleration of the Standard 2D task car. This variant will be used to show how transfer efficacy changes when the source task transition function changes.

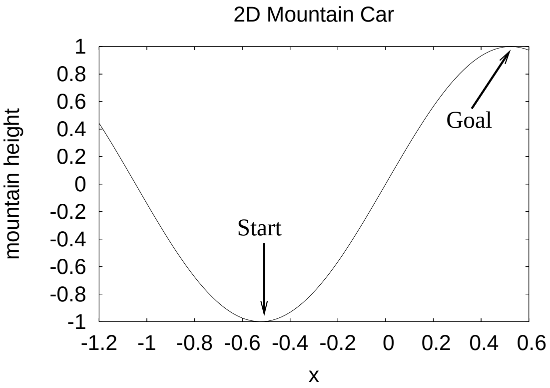


Fig. 1. In the standard 2D mountain car the agent must travel along a curve (mountain).

4.2 Three Dimensional Mountain Car

The 3D task [23] extends the mountain's curve into a surface (see Figure 2).[5] 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 -0.0007 and +0.0007 respectively, while South and North modify $\dot{y}$ by -0.0007 and +0.0007 respectively.[6] 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 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 spatial dimensions at any given time, one cannot simply "factor" this problem into the simpler 2D task. While data gathered from the 2D task 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] In the original formulation, the velocity was changed by ± 0.001 due to acceleration. We have reduced the power of the car to make the task more challenging.

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

[5] An animation of a trajectory from a trained policy can be found at `http://www.cs.utexas.edu/~mtaylor/3dMtnCar.html`.

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

4.3 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 [15]. 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 (which determines the model's resolution). For any state $\mathtt{x} \in X$ and action $a \in A$, Fitted R-MAX estimates $T(\mathtt{x}, a)$ using a probability distribution over instances (s_i, a, r_i, s_i') in the data available for action a. Each instance i is given a weight w_i depending on the Euclidean distance from $\mathtt{x}$ to s_i and on the *model breadth* parameter b, according to the following formula: $w_i \propto e^{-\left(\frac{|\mathtt{x}-s_i|}{b}\right)^2}$. Intuitively, b controls the degree of generalization used to estimate $T(\mathtt{x}, a)$ from nearby data. In 3D mountain car experiments, we used a parameter of $b = 0.4$. In theory, all instances that share the action a could be used to help approximate $\mathtt{x}$, where each instance i's contribution is modified by w_i (i.e., a Gaussian weighting that exponentially penalizes distance from $\mathtt{x}$). To reduce the computational cost of the algorithm, for a given state $\mathtt{x}$ we computed the weights for the nearest instances first. Once an instance's weight failed to increase the cumulative weight by at least 10%, we ignored the remaining instances' contribution as negligible (the *minFraction* parameter in Algorithm 1). 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.

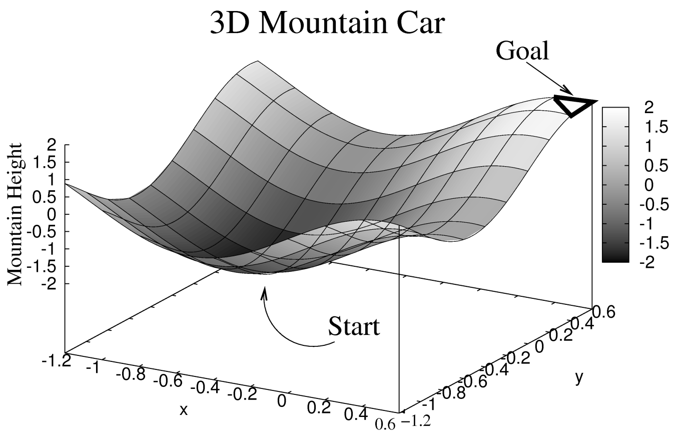


Fig. 2. In 3D mountain car the 2D curve becomes a 3D surface. The agent starts at the bottom of the hill with no kinetic energy and attempts to reach the goal area in the Northeast corner.

Changing the learning parameters for Fitted R-MAX outlined above affects 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

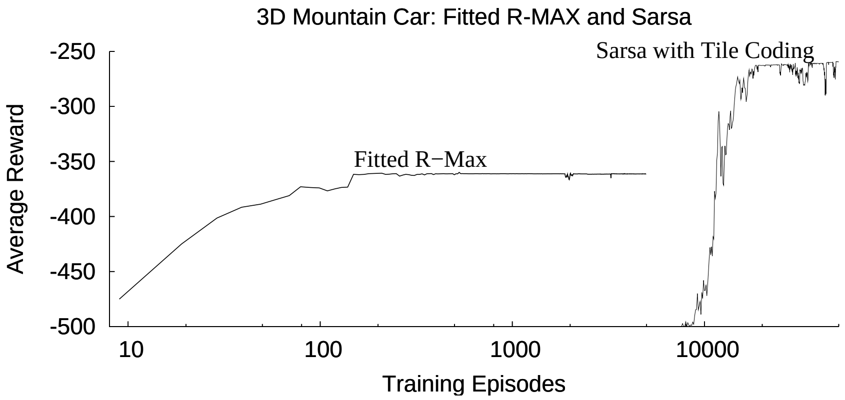


Fig. 3. Average learning curves for Fitted R-MAX and Sarsa show the significant speed advantage of model-based RL on the 3D mountain car task (note the log scale). Fitted R-MAX parameters where chosen for relatively low sample and computational complexity requirements at the expense of asymptotic performance.

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.

Figure 3 compares the average performance of 12 Fitted R-MAX trials with 12 Sarsa trials in the 3D mountain car task. The ϵ-greedy Sarsa(λ) agent uses a CMAC [24] function approximator with 14 4-dimensional linear tilings, which is analogous to how Singh and Sutton [12] used 14 2-d dimensional linear tile codings for their 2D task.[7] This result demonstrates that Fitted R-MAX can be tuned so that it learns with significantly less data (finding a path to the goal in roughly 50 episodes instead of 10,000 episodes), but does not necessarily achieve optimal performance. Learning with Fitted R-MAX takes substantially more computational resources than Sarsa in this domain; the Fitted R-MAX learning curves were terminated once their performance plateaued (and thus are run for fewer episodes than Sarsa).

5 TIMBREL Implementation for Mountain Car

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,

[7] We achieved the best performance on this task by setting the learning rate to $\alpha = 0.5$, the exploration rate to $\epsilon = 0.1$, $\lambda = 0.95$, not decaying α, and decaying ϵ at the end of every episode by 0.1%.

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 our past work [23] has demonstrated that a mapping between the 2D and 3D mountain car tasks may be learned autonomously. 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 $\mathtt{x}$ that we wish to approximate.

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.

As discussed in Section 2, Fitted R-MAX approximates transitions from a set of sample states $\mathtt{x} \in X$ 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 $T(\mathtt{x}_{\mathbb{T}}, a_{\mathbb{T}})$ for any $\mathtt{x}$ 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 $T(\mathtt{x}_{\mathbb{T}}, a_{\mathbb{T}})$.

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 $T(\mathtt{x}_{\mathbb{T}}, a_{\mathbb{T}})$. 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 in Algorithm 2). 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 9), and add it to the data approximating T (line 10).

As an illustrative example, consider the case when the agent wants to approximate $T(\mathtt{x}_{\mathbb{T}}, a_{\mathbb{T}})$, where $\mathtt{x}_{\mathbb{T}} = \langle x_{\mathbb{T}}, y_{\mathbb{T}}, \dot{x}_{\mathbb{T}}, \dot{y}_{\mathbb{T}} \rangle = \langle -0.6, -0.2, 0, 0.1 \rangle$ 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 defined mappings for 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 $\mathtt{x}_{\mathbb{T}}$ we wish to approximate. Instead, we recognize that this sample was selected from the source task to be near to $\mathtt{x}_{\mathbb{T}}$ 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

$$(\langle -0.61, y_{\mathbb{T}}, 0.01, \dot{y}_{\mathbb{T}} \rangle, \text{East}, -1, \langle -0.59, y_{\mathbb{T}}, 0.2, \dot{y}_{\mathbb{T}} \rangle) =$$
$$(\langle -0.61, -0.2, 0.01, 0.1 \rangle, \text{East}, -1, \langle -0.59, -0.2, 0.02, 0.1 \rangle).$$

The analogous step is then performed for the second selected source task tuple: the source task tuple is transformed with $\mathcal{X}$, and $x_{\mathbb{T}}$ and $\dot{x}_{\mathbb{T}}$ are held constant. Finally, both transferred instances are added to the $T(\mathtt{x}, a)$ approximation.

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 $T(\mathtt{x}_{\mathbb{T}}, a_{\mathbb{T}})$ for different values of $(\mathtt{x}, a)$ 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 $\mathcal{X}$). 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 4.3, Fitted R-MAX uses the distance between instances and $\mathtt{x}$ to calculate instance weights. When an instance is used to approximate $\mathtt{x}$, 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 $(\mathtt{x}, a)$ 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 $\mathtt{x}$) does not exceed the total weight threshold, source task data is transferred to allow an approximation of $T(\mathtt{x}_{\mathbb{T}}, a_{\mathbb{T}})$. 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 $\mathtt{x}$ does not reach 1.0, R_{max} is assigned to the model for $\mathtt{x}$.

As a final implementation note, consider what happens when some $\mathtt{x}$ 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 $\mathtt{x}$). 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 4.3.

6 Transfer Experiments

To test the efficacy of TIMBREL, we first conducted an experiment to measure the learning speed of Fitted R-MAX in the mountain car domain with and without TIMBREL. Roughly 50 different sets of Fitted R-MAX parameters were used in preliminary experiments to select the best settings for learning the 3D task without transfer (as discussed in Section 4.3). We ran 12 trials for 4,000 episodes and found that 10 out of 12 trials were able to converge to a policy that found the goal area. Recall that Fitted R-MAX is not guaranteed to converge to an optimal policy because it depends on approximation in a continuous state space.

To transfer from the Standard 2D mountain car task into the more complex 3D mountain car, we first allow 12 Fitted R-MAX agents to train for 100 episodes each in the 2D task while recording all observed (s, a, r, s') transitions.[8] We then used TIMBREL to train agents in the target task for 1,000 episodes. 12 out of 12 trials converged to a policy that found the goal area.

After learning, we averaged over all trials for the non-transfer and transfer learning trials. For clarity, we also smoothed the curves by averaging over a 10 episode window. Figure 4(a) shows the first 1000 episodes of training (running the experiments longer than 1,000 episodes did not significantly improve the policy, as suggested by Figure 3). T-tests determined that all the differences in the averages were statistically significant ($p < 0.05$), with the exception of the initial average at episode 9. This result confirms that transfer can significantly improve the performance of agents in the 3D mountain car task.

We hypothesize that the U-shaped transfer learning curve is caused by a group of agents that find an initial path to the goal, spend some number of episodes exploring to find a faster path to the goal, and ultimately return to the original policy (see Figure 4(b)). In addition to improved initial performance, the asymptotic performance is improved, in part because some of the non-transfer tasks failed to successfully locate the goal. The difference in success rates (10 of 12 trials reaching the goal vs. 12 of 12) suggest that transfer may make difficult problems more tractable.

TIMBREL, and its implementation, were designed to minimize 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.[9] 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 of increasing computational requirements and reducing sample complexity is highly advantageous, and is one of the benefits inherent to model-based reinforcement learning.

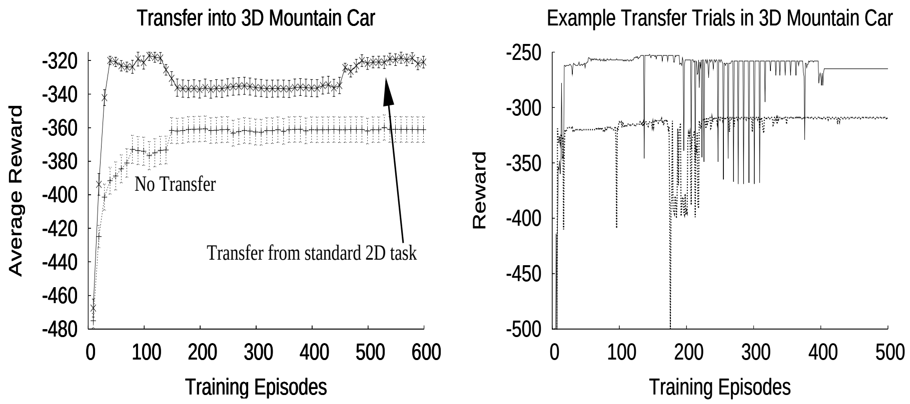
Our second experiment examines how the amount of recorded source task data affects transfer. One hypothesis was that more tuples in the source task would equate to higher performance in the target task, because the target task agent would have more data to draw from, and thus would be better able to approximate any given $T(\mathtt{x}, a)$.

We first ran experiments in the Standard 2D task for 5, 10, and 20 episodes, where the average number of steps per episode was 422, 298, 238, respectively.[10] Figure 5(a)

[8] We experimented with roughly 10 different parameter settings for Fitted R-MAX in the Standard 2D task. Every episode lasts 500 time steps if the goal is not found and the 2D goal state can be reached in roughly 150 time steps. When learning 2D Mountain car, the agent experienced an average of 24,480 source task transitions during the 100 source task episodes.

[9] An analysis of the increase in computational complexity depends on the amount of data transferred into a target task, which in turn depends on the pair of tasks used.

[10] The average number of steps per episode decreases for longer trials because the agent quickly learns to find the goal.



(a) **(b)**

Fig. 4. (a) 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. **(b)** As Fitted R-MAX explores, the performance can vary significantly, sometimes resulting in a U-shaped learning curve.

shows that transfer from 20 source task episodes is similar to using 100 source task episodes and performs statistically better than no transfer at the 95% level for 98 of the 100 points graphed. While transfer performance degrades for trials that use 10 and 5 source task episodes, both trials do show a statistically significant boost to the agents' *initial* learning performance. This result demonstrates that a significant amount of information can be learned in just a few source task episodes; the source task is less complex than the target and thus a short amount of time spent learning in the source may have a large impact on the target task performance.

Recall the mountain car has a reward of -1 on each time step. The agent learns to reach the goal area because transitioning into this area ends the episode and the steady stream of negative reward. The third experiment uses the No Goal 2D task as a source task to examine how changing the reward function in the source task affects transfer. When training in the source task, every episode lasted 500 time steps (the maximum number of steps). After learning for 100 episodes in the source task, we transferred into the target task and found that 9 of the 12 trials successfully discovered policies to reach the goal area. Figure 5(b) suggests that transfer from a source task policy with a different reward structure can be initially useful (t-tests confirm that transfer outperforms non-transfer for four of the first five points graphed), but the relative performance of the non-transfer trials soon outperform that of learning with transfer.

Our fourth experiment uses the High Power 2D task as a source task. We again record 100 episodes worth of data for source task learners and use TIMBREL to transfer into 3D mountain car. Because the source task uses a car with a motor more than twice as powerful as in the 3D task, the transition function learned in the source task is less useful to the agent in the target task. 9 of the 12 target task trials successfully converged to a policy that reached the goal. Transferring from the High Power 2D task (Figure 5(b)) is not as useful as transferring from the Standard 2D Task (Figure 5(a)) due to differences in the transition functions. Although t-tests show that there is a statistically significant improvement at the beginning of learning, the transfer and non-transfer curves in Figure 5(b) quickly become statistically indistinct with more target task training.

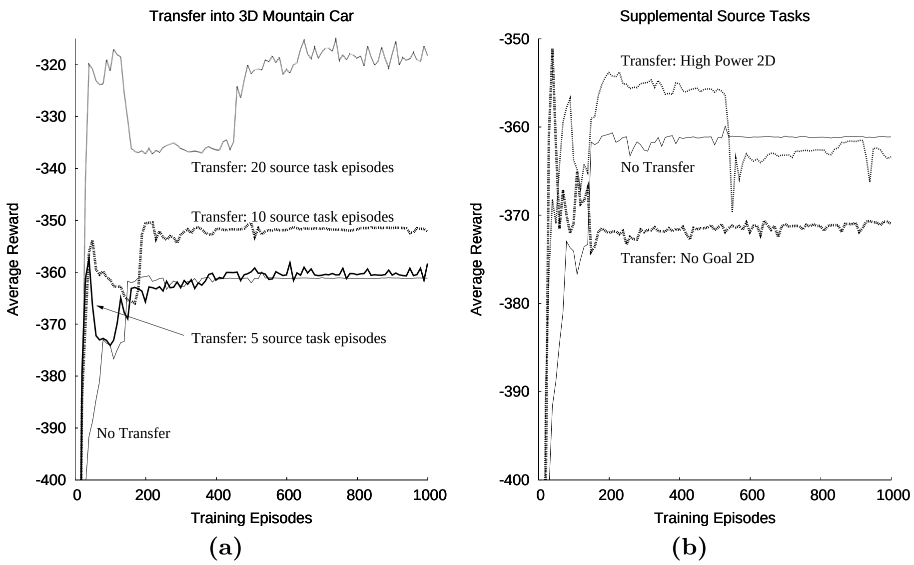


Fig. 5. (a) This graph shows the effect of different amounts of source task training. Each learning curve is the average of 12 independent trials. **(b)** Transfer from a 2D mountain car task that has no goal state or from a 2D mountain car with significantly stronger acceleration produces statistically significant improvements at the beginning of learning when compare to learning without transfer. However, this relative advantage is lost as the target task agents gain more experience.

Figure 5(b) highlights an important drawback of transfer learning. Transfer efficacy is often affected by the similarity of source tasks and target tasks, and in some circumstances transfer may not help the learner. Indeed, other experiments (not shown) confirm that if T or R in the source and target tasks are too dissimilar, transfer may actually cause the learner to learn more slowly than if it had not used transfer. While there is not yet a general solution to avoiding *negative transfer*, our recent results [23] suggest that the "relatedness" of tasks may be possible to measure empirically, and may guide learners when deciding whether or not to transfer.

7 Conclusion and Future Work

In this paper we have introduced TIMBREL, a transfer method fully 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 future research directions suggested by this work. It would be informative to study how transfer efficacy changes when the amount of exploration is changed in the source task. Put differently, if the agent has 100 episodes to learn the source task, can it intelligently set its parameters to maximize transfer efficacy? In our experiments we used the default threshold value of 1.0 to determine if a particular approximation of $T(\mathbf{x}_{\mathbb{T}}, a_{\mathbb{T}})$ has enough data. This is related to the amount of exploration and its value may impact the efficacy of transfer, but tuning this parameter is left to future work.

All parameters for Fitted R-MAX were tuned when learning without transfer. It is possible that the model breadth parameter, b, may change the efficacy of transfer. Section 5 specified that a maximum of 20 source task instances were used to approximate a single target task transition. This parameter was set during initial experimentation, but further tuning could improve transfer performance. Lastly, none of the experiments using Fitted R-MAX attained an asymptotic performance equivalent to Sarsa (Figure 3). It may be worth re-tuning the base learning algorithm's parameters to maximize asymptotic performance (at the expense of computational and sample complexity), and then determine if TIMBREL can compensate so that learning experiments terminate in a reasonable amount of time.

We predict that TIMBREL will work, possibly with minor modifications, in other model-based RL algorithms. For instance, TIMBREL could be directly used in the planning phase of *Dyna-Q* [1] as a source of simulated experience when the agent's model is poor (such as at the beginning of learning a target task). TIMBREL should also be useful, without modification, in R-MAX.

Lastly, we intend to apply TIMBREL in more complex domains with continuous state spaces (which may show relatively more benefit from transfer than the simple tasks discussed in Section 6 [6]). Although this paper focuses on tasks in the mountain car domain, Algorithm 2 is applicable in many settings. Future work to empirically determine how well TIMBREL functions in other domains, and when applied to pairs of tasks with qualitative differences not explored in this paper, will help to better understand and quantify the benefits that instance transfer provides.

Acknowledgments

We would like to thank Lilyana Mihalkova, Joseph Reisinger, Raymond J. Mooney, and 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.

References

1. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press (1998)
2. Tesauro, G.: TD-Gammon, a self-teaching backgammon program, achieves master-level play. Neural Computation **6**(2) (1994) 215–219
3. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.: Inverted autonomous helicopter flight via reinforcement learning. In: International Symposium on Experimental Robotics. (2004)
4. Kohl, N., Stone, P.: Machine learning for fast quadrupedal locomotion. In: The Nineteenth National Conference on Artificial Intelligence. (July 2004) 611–616
5. Torrey, L., Walker, T., Shavlik, J., Maclin, R.: Using advice to transfer knowledge acquired in one reinforcement learning task to another. In: Proceedings of the Sixteenth European Conference on Machine Learning. (2005)
6. Taylor, M.E., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. Journal of Machine Learning Research **8**(1) (2007) 2125–2167

7. Lazaric, A., Restelli, M., Bonarini, A.: Transfer of samples in batch reinforcement learning. In: Proceedings of the 25th Annual ICML. (2008) 544–551
8. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**(1) (1996) 69–101
9. Caruana, R.: Multitask learning. Machine Learning **28** (1997) 41–75
10. Wilson, A., Fern, A., Ray, S., Tadepalli, P.: Multi-task reinforcement learning: a hierarchical bayesian approach. In: ICML '07: Proceedings of the 24th international conference on Machine learning, New York, NY, USA, ACM Press (2007) 1015–1022
11. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, UK (1989)
12. Singh, S., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Machine Learning **22** (1996) 123–158
13. Ng, A.Y., Jordan, M.: PEGASUS: A policy search method for large MDPs and POMDPs. In: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence. (2000)
14. Brafman, R.I., Tennenholtz, M.: R-Max – a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research **3** (2002) 213–231
15. Jong, N.K., Stone, P.: Model-based exploration in continuous state spaces. In: The Seventh Symposium on Abstraction, Reformulation, and Approximation. (July 2007)
16. Bellman, R.E.: Dynamic Programming. Princeton University Press (1957)
17. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc. (1994)
18. Atkeson, C.G., Santamaria, J.C.: A comparison of direct and model-based reinforcement learning. In: Proceedings of the 1997 International Conference on Robotics and Automation. (1997)
19. Tanaka, F., Yamamura, M.: Multitask reinforcement learning on the distribution of MDPs. Transactions of the Institute of Electrical Engineers of Japan. C **123**(5) (2003) 1004–1011
20. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning **13** (1993) 103–130
21. Dearden, R., Friedman, N., Andre, D.: Model based bayesian exploration. In: Proceedings of the 1999 conference on Uncertainty in AI. (1999) 150–159
22. Sunmola, F.T., Wyatt, J.L.: Model transfer for Markov decision tasks via parameter matching. In: Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006). (December 2006)
23. Taylor, M.E., Kuhlmann, G., Stone, P.: Autonomous transfer for reinforcement learning. In: The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems. (May 2008)
24. Albus, J.S.: Brains, Behavior, and Robotics. Byte Books, Peterborough, NH (1981)