In *The Autonomous Agents and Multi-Agent Systems Conference* (AAMAS-07),
Estoril, Portugal, May 2008.

Autonomous Transfer for Reinforcement Learning

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

ABSTRACT

Recent work in *transfer learning* has succeeded in making *reinforcement learning* algorithms more efficient by incorporating knowledge from previous tasks. However, such methods typically must be provided either a full model of the tasks or an explicit relation mapping one task into the other. An autonomous agent may not have access to such high-level information, but would be able to analyze its experience to find similarities between tasks. In this paper we introduce *Modeling Approximate State Transitions by Exploiting Regression* (MASTER), a method for automatically learning a mapping from one task to another through an agent's experience. We empirically demonstrate that such learned relationships can significantly improve the speed of a reinforcement learning algorithm in a series of Mountain Car tasks. Additionally, we demonstrate that our method may also assist with the difficult problem of task selection for transfer.

Keywords

Transfer Learning, Reinforcement Learning

1. INTRODUCTION

Agents deployed in an environment often need to learn how to execute sequential actions. A common way of framing such problems is to use the framework of *reinforcement learning* [17] (RL). While RL algorithms have had many empirical success and have some theoretical guarantees, for RL to be widely applicable on real-world tasks, it is important for learning to occur with as little training experience as possible. If RL algorithms can learn new tasks from limited experience, agents may be able to learn reliably on-line in the real world. One approach to enabling such learning is to employ *transfer learning* (TL) to reuse knowledge gathered in previous tasks to learn a novel task better or faster.

A number of recent empirical successes (e.g., [4, 11, 20, 22]) in a variety of RL domains have shown that transfer between RL tasks is feasible. Such successes are not entirely surprising due to the intuitive appeal of transfer. If a learning agent experiences two or more similar tasks, we would expect it to be possible for the agent to leverage past knowledge; we have such an existence proof in human learning.

Current TL algorithms are able to successfully transfer knowledge from one or more *source tasks* into a novel *target task*. However, as discussed in the next section, existing algorithms typically need the relationship between the source and target tasks to be specified by a human. While effective in some situations, TL methods that rely on being provided such information will be unable to transfer knowledge autonomously.

If TL methods are able to automatically learn task relationships, transfer may be possible in domains where humans are unable to intuit accurate inter-task mappings. In order to enable transfer in such an autonomous transfer learning agent, a TL algorithm must:

1. Select an appropriate source task from which to transfer, given a target task.
2. Learn how the source task and target task are related.
3. Effectively transfer knowledge from the source task to the target task.

While significant progress has recently been made on step three, relatively little attention has been paid to the first two. This paper introduces *Modeling Approximate State Transitions by Exploiting Regression* (MASTER). To the best of our knowledge, MASTER is the first TL method able to autonomously learn a relationship between two tasks by using experience gathered from task environments, rather than from human-provided environmental models. We empirically verify our method by learning mappings between different tasks in the Mountain Car domain and demonstrate a significant improvement in training, relative to learning without transfer.

2. BACKGROUND AND RELATED WORK

In this section we discuss related work to emphasize how MASTER is situated in the current literature. We first briefly discuss the *Markov decision process* (MDP) framework that is commonly used to describe RL tasks. An agent observes the world through some sensors and describes its current *state* $s \in S$ that can be decomposed into a sequence of *state variables*, $s = \langle x_1, x_2, \ldots, x_k \rangle$. The agent begins in s_{initial} and, if the task is episodic, the task terminates if the agent reaches s_{final} (or, in general, any state in a set of final states). The agent may execute an action $a \in A$, that is possibly state-dependent. The next state the agent arrives at is governed by a transition function, $T : S \times A \mapsto S$, which may be stochastic. Upon arriving at a new state, the agent receives an immediate reward $R : S \mapsto \mathbb{R}$. The goal of an agent is to learn a policy $\pi : S \mapsto A$ such that the expected long-term reward is maximized. A value function learner estimates the value function $V : S \mapsto \mathbb{R}$ or the action-value function $Q : S \times A \mapsto \mathbb{R}$ from experience, rather than learning the policy directly.

Many existing method have recently been developed that allow transfer between pairs of RL tasks. For instance, Torrey et al. [22] use advice learned in a source task to speed up learning in a target task. Taylor, Stone, and Liu [20] leverage a learned action-value function to learn a target task faster. Both of these methods assume a hand-coded *inter-task mapping*, which defines how the state variables in both tasks are related, and how actions in both tasks are

Cite as: Autonomous Transfer for Reinforcement Learning, M. E. Taylor, G. Kuhlmann, and P. Stone, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal.

related. Many current TL methods differ primarily in the type of knowledge saved in the source task, the form of the inter-task mapping, and the way the inter-task mapping is used to make the saved source task knowledge useful in the target task.

Most closely related to this work are approaches for transfer learning in RL domains that are able to learn an inter-task mapping for pairs of tasks. Liu and Stone [7] assume that the agent is provided a complete and correct transition model for both the source task and the target task. They are then able to use a graph-matching algorithm that finds similarities between state variables in the two tasks, and actions in the two tasks. Kuhlmann and Stone [6] approach a similar problem in the *General Game Playing* [5] domain. They construct *rule graphs* based on the provided transition and reward functions for the source and target tasks, and then find a match between games based on graph similarity.

A different set of work has attempted to reduce the amount of information needed for the agent to learn a mapping from a source to target task. Soni and Singh [15] treat the different possible state variable mappings as *options* (multi-step actions) in the target task and use them to learn the target task faster. However, this work assumes that the action mapping is provided to the agent and that the state variables can be grouped into task-independent groups. This assumption allows the source and target tasks to have a different number of objects in the different tasks because each object can be described by the same number of state variables. Talvite and Singh [18] again assume an action mapping and state variable grouping and use an experts algorithm to select between the different possible state variable mappings. Taylor, Whiteson, and Stone [21] relax the requirements somewhat by learning both the state variable mapping and the action mapping by using classification, but they also leverage the assumption that the agent is provided state variable groupings.

Other work attempts to bypass the need for an inter-task mapping altogether. If the problem is formulated in the *relational reinforcement learning* [3] (RRL) framework, source task policies can be directly reused if the number of objects change in subsequent tasks. For instance, transfer can be quite effective [11] in the Blocksworld domain when source and target tasks have different numbers of blocks. However, not all tasks can easily be formulated as relational problems, which includes the tasks we use in this work (see Section 4).

Lastly, there has been some work in effectively selecting a source task from which to transfer, given a target task. Fernandez and Veloso [4] first construct a library of learned policies. When a novel task is experienced, the agent can learn to probabilistically exploit the policies in the library. While no explicit inter-task mapping is needed, some time must be spent in the target task trying to determine which policy to use, and the amount of experience required to select the best mapping will increase with the library size. Additional constraints include restricting the reward function to a single goal state and the transition function to remain unchanged in the different tasks. Ultimately, it is likely that a *case based reasoning* [1] (CBR) approach may be successfully used to find task similarities between a target task and previously learned tasks. While there have been initial attempts at using CBR to assist with transfer [13], we are not aware of a robust, a domain-independent similarity metric for MDPs.

3. THE MASTER METHOD

As discussed previously, many TL methods rely on a mapping between the source task and target task to enable transfer. Such a mapping is typically provided to the learner by an oracle or can be determined by analyzing models provided for both tasks. In this section we introduce MASTER, our method for learning an inter-task mapping from environmental data. We can learn both the action mapping, χ_A, and the state variable mapping, χ_X, from data collected in the source and target tasks. χ_A and χ_X fully define an inter-task mapping, which maps each target task action into one or more source task actions, and maps each target task state variable into one or more source task state variables. Implementation-level details will be specified later in the context of specific transfer experiments.

Method 1 MASTER

1: **while** training in the source task **do**
2: Agent(s) record observed (s, a, s') tuples in D_{source}
3: Save learned knowledge
4: **for** small number of episodes in the target task **do**
5: Agent(s) record observed (s, a, s') tuples in D_{target}
6: Learn a one-step transition model, $M_{target}(s, a) \mapsto s'$, that tries to minimize$\sum_{D_{target}}(M_{target}(s, a) - s')^2$
7: **for** every possible 1-to-1 mapping from source task state variables to target task state variables, δ_X **do**
8: **for** every possible 1-to-1 mapping from source task actions to target task actions, δ_A **do**
9: Use δ_X and δ_A to transform D_{source} into D'_{source}
10: **for** every tuple $(s, a, s') \in D'_{source}$ **do**
11: Calculate the error: $(s' - M_{target}(s, a))^2$
12: $MSE_{(\delta_X, \delta_A)} \leftarrow$ average error
13: Use the recorded MSE values to construct χ_A and χ_X from some δ_A^{-1} and δ_X^{-1}

Our domain-independent method for constructing inter-task mappings is summarized in Method 1. We consider five distinct phases:

1. Lines 1–3 represent training in the source task. Any learning method can be used that is capable of utilizing inter-task mappings for transfer (e.g., KBKR [8], Sarsa [12, 14], or NEAT [16]). The type of knowledge saved in the data structure D_{source} will depend on which RL algorithm is used for source task learning.
2. Lines 4–5 show the agent(s) exploring in the target task without learning. We have found in practice that only a relatively small amount of data is needed (see Section 5.3).
3. A one-step transition model, M_{target}, for the target task is learned on line 6. As discussed in Section 5, our experiments utilize neural network function approximation in the Weka [23] machine learning package, but we expect other prediction methods to also perform well. Note that the error calculation $(M_{target}(s, a) - s')$ is a vector operation and is computed per state variable. Such an error definition implicitly assumes that the state variables can be scaled so that they are weighted equally, and that a Euclidean metric is an appropriate measure of state similarity (for both discrete and continuous state variables).
4. Lines 7–12 examine different ways of mapping the source task data into the target task using inter-task mappings (δ_X and δ_A). When considering target tasks that have more state variables and/or actions than the source task, this is typically a one-to-many mapping. Each possible mapping is tested and its appropriateness is determined by how well it matches the learned model.
5. Lastly, the agent constructs the inter-task mapping from the tested mappings (line 13). Note that the inter-task mapping maps target task data to source task data, while the agent had been testing different mappings from source task data into the target task. Details of this step will be discussed in Section 5, but the intuition is that if a there is a single best mapping, it should be used. If there are a number of candidate mappings that have very similar MSEs, they can be combined in a mixture weighted by their inverse errors.

After MASTER has determined the inter-task mappings, they can be leveraged in conjunction with the saved knowledge (Line 3) to speed up learning in the target task using one of the existing TL methods for RL tasks.

The key insight of this method is that it is able to propose all possible methods and then score them by analyzing them off-line (i.e., without requiring more samples from the environments). Such analysis, lines 7–12, is exponential in the number of state variables and actions. While such testing is relatively fast, if this method is scaled to tasks with a large number of state variables or actions, some type of heuristic will need to be used. For instance, rather than an exhaustive search, a hill-climbing method could be used to find a good mapping (for instance, a variant on Powell's Method [10]). Additionally, it is worth emphasizing that this search affects only computational complexity. In this work we attempt to learn an inter-task mapping so that the sample complexity of the target task is reduced – reducing the computational complexity is not our primary concern, as CPU cycles are generally cheap when compared to collecting data from a fielded agent.

There are a number of model-learning methods for RL tasks (e.g., KBRL [9]), but such methods do not generally scale to large tasks with continuous state variables, which are of particular interest to agents acting in real-world tasks. Such methods generally attempt to model a task in order to perform dynamic programming offline. In MASTER, we instead only need to learn an *approximate* model that allows us to find similarities between state variables and actions in two tasks. Since we typically would expect relatively large differences in the transition model of an MDP when state variables and actions are changed, the error due to poor modeling is less critical then when a model is used for dynamic programming. Furthermore, existing model-learning methods generally do not scale well to large, continuous state spaces. This relaxed requirement allows us to use a simple regression method, which may be used on tasks with continuous state variables, and which requires relatively little data for model learning. [1]

4. GENERALIZED MOUNTAIN CAR

In this section we introduce our experimental domain, a generalized version of Mountain Car [14], and summarize how tasks in this domain are learned without the aid of transfer. 2D Mountain Car is one of the canonical RL tasks which requires generalization across a continuous state space where an agent must drive an underpowered car up a mountain to reach a goal state. We then extend the problem to three dimensions. This extension retains much of the structure of the 2D problem so that transfer from 2D to 3D may be beneficial, but the 3D task forces the agent to act in a state space with four continuous state variables instead of only two. Additionally, we later allow the agent to execute a new action (engage the hand brake). By adding this action we are able to experiment on a total of four related tasks, each with different state variables and actions. After we have introduced the tasks, Section 5 discusses how MASTER is able to learn inter-task mappings for these tasks and demonstrates their benefit by using an existing TL algorithm.

In all the mountain car tasks, the shape of the mountain and the goal location are initially unknown. The agent begins at rest at the bottom of the hill. The reward is -1 for each time step until the goal is reached, at which point the episode ends. The episode also ends, and the agent is reset to the start state, if the agent fails to find the goal within 5000 time steps.

[1]If a significant amount of data from the target task were needed to learn a transition model, relative to the amount of data needed to learn the target task, the time spent gathering data to learn an inter-task mapping could easily outweigh any savings gained by transfer.

4.1 Two Dimensional Mountain Car

In the standard two dimensional version of Mountain Car, the agent's state is described by two continuous state variables: horizontal position (x) and velocity ($\dot{x}$), which are restricted to the ranges $[-1.2, 0.6]$ and $[-0.07, 0.07]$ respectively. The agent has three actions: {`Left`, `Neutral`, `Right`}, which change the velocity by -0.001, 0, and 0.001 respectively. On each time step the term $-0.025(cos(3x))$ is added to $\dot{x}$ to account for gravity. The goal state is $x = 0.5$, without regard to the current velocity. We use a released version of this code for our simulations. [2]

Our agent uses Sarsa(λ) [12] with CMAC [2] (tile coding) function approximation. The CMAC is two-dimensional and has 14 tilings (repeating the setup detailed in Singh and Sutton [14]). Sarsa has learning rate of $\alpha = 0.5$, an ϵ-greedy exploration rate of $\epsilon = 0.1$, an eligibility trace decay rate λ of 0.95, and we multiply the exploration rate by 0.99 at the end of each learning episode to assist convergence. The learning rate is not decayed. These settings were selected because they were included in the released Mountain Car package as the best found to date for Sarsa(λ) on this task.

4.2 Three Dimensional Mountain Car

In this novel modification to the standard Mountain Car domain, the mountain's curve is extended to a 3D surface (see Figure 1). The state now has four continuous state variables: $x, \dot{x}, y, \dot{y}$. The positions have a range of $[-1.2, 0.6]$ and the velocities are constrained to $[-0.07, 0.07]$. The agent now selects from five actions: {`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.[3] On each time step $\dot{x}$ is updated by $-0.025(cos(3x))$ and $\dot{y}$ is updated by $-0.025(cos(3y))$ due to gravity. The goal state is $x \geq 0.5, y \geq 0.5$. When learning this task without transfer we use a four-dimensional CMAC with 14 tilings, and again set $\lambda = 0.95$. After initial experiments with roughly 100 different parameter settings, we selected $\alpha = 0.2$, $\epsilon = 0.5$, and an ϵ-decay of 0.99.

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

[3]Note that we call the agent's vehicle a "car," although it does not turn, to emphasize the similarities with the stand two dimensional mountain car task.

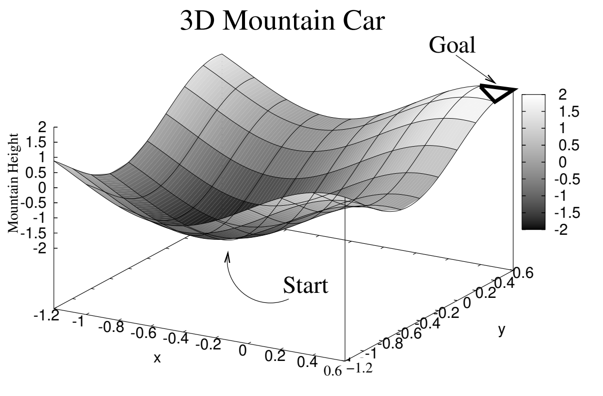

Figure 1: In 3D Mountain Car, the 2D curve becomes a 3D surface.

4.3 Hand Brake Mountain Car

We add a variant to the two- and three-dimensional mountain car tasks by adding an extra action: `hand brake`. While the other actions in the mountain car tasks are all executed for a single time step, the hand brake action is a macro-action that executes for five simulator time steps. The effect of the action, in both the two- and three-dimensional versions of the task, is to immediately set the velocity of the car to zero. All other aspects of the tasks remain unchanged from the non-hand brake versions.

When learning the two-dimensional hand brake mountain car we used parameters identical to the two-dimensional mountain car. When learning the three-dimensional hand brake mountain car without transfer we tested roughly 70 different parameter settings and selected $\alpha = 0.4$, $\epsilon = 0.3$, and an ϵ-decay of 0.99.

5. EXPERIMENTAL VERIFICATION

In this section we show how MASTER can learn an inter-task mapping from 2D Mountain Car to 3D Mountain Car. We then discuss a number of experiments that illustrate how our method is able to achieve a significant speed-up in the target task with limited source task data and compare the results of our algorithm with an existing learning approach. Lastly, we demonstrate how MASTER can evaluate mappings between multiple source tasks and help to select an appropriate source task for transfer.

5.1 Using MASTER in Mountain Car

In order to use MASTER to learn an inter-task mapping between 2D Mountain Car and 3D Mountain Car, the agent first trained in 2D Mountain Car for 100 episodes using Sarsa(λ) while saving the observed (s_s, a_s, s'_s) transitions (see Method 1, lines 1–3). The agent then executed actions randomly in 3D Mountain Car for 50 episodes, recording the observed (s_t, a_t, s'_t) transitions.

To learn the one-step transition model (Method 1, line 6), we used the Weka package (version 3.4.6) to train multi-layer perceptrons (i.e., artificial neural networks). While we experimented primarily with neural networks for building the 1-step model, we expect that other non-linear function approximators could work equally well. After trying 4 different parameter settings in informal experiments, we used Weka's default settings, except for setting the number of hidden nodes to eight and the number of training epochs to 5000. For each 3D Mountain Car trail, we trained a separate neural network for each (action, state variable) pair, resulting in a total of 20 trained neural networks. Each network modeling the target task data for 3D Mountain car had 4 inputs, one for each state variable, eight hidden nodes, and a single output that would predict a single state variable's next value.

The trained target task models are only approximate because of the small amount of target task data. For instance, when we compare trajectories in the target task with trajectories produced by our model (see Figure 2), it is clear that the models are not very accurate, but that the relative effects of actions are preserved.

Once the models are learned, the agent next iterates over all possible state variable and action mappings. For instance, it would sequentially try mapping x in the source task to each of $\{x, y, \dot{x}, \dot{y}\}$ in the target task. Likewise, the source action Left would be mapped to the target actions {Neutral, West, East, South, North}. The agent then transforms the recorded 2D Mountain Car data using each of these 240 mappings (16 state variable mappings × 15 action mappings). For instance, consider a recorded source task tuple $(x, \dot{x}, \text{Left})$, the state variable mapping $x_s \mapsto \{x_t, y_t\}$, $\dot{x}_s \mapsto \{\dot{x}_t, \dot{y}_t\}$, and the action mapping $\text{Left}_s \mapsto \text{West}_t$, $\text{Right}_s \mapsto \{\text{Neutral}_t, \text{East}_t, \text{South}_t, \text{North}_t\}$. Using these mappings, the tuple will be transformed into $(x, x, \dot{x}, \dot{x}, \text{West})$. Each transformed tuple is used as input to the neural networks for the relevant target task action. In our example, we would use the set of four neural networks trained on the target task action West to predict the next state that the agent observes. The output from each neural network is compared with the true next state the agent observed in the source task, and the error over all the transformed source task data is used to calculate the MSE for the mapping.

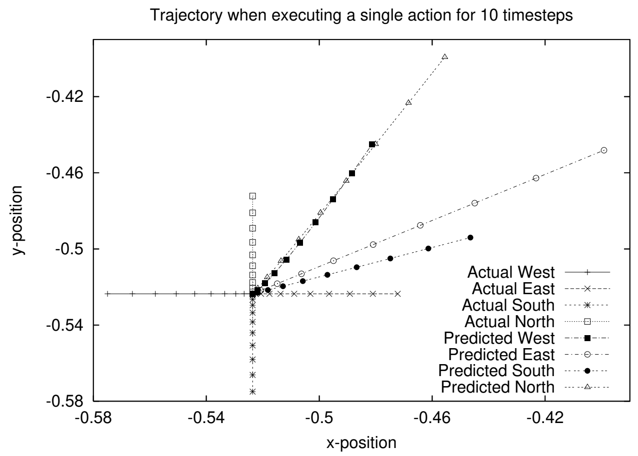

Figure 2: Trajectories in the 3D Mountain Car task (10 "Actual" actions are taken in a row), and trajectories generated by neural networks trained on 50 target task episodes (10 "Predicted" actions are taken in a row), shows how the trained neural network may produce skewed predictions, but that the relative effect of the actions is preserved.

Table 1 summarizes results of a representative trial when evaluating the 240 mappings. For this domain, one state variable mapping is significantly better than all others, both when averaged across all action mappings, or when the best action mapping is considered for each possible state variable mapping. This state variable mapping is fairly intuitive: the position state variable in the 2D Mountain Car maps to both position variables in 3D Mountain Car, and the velocity state variable in the 2D Mountain Car maps to both velocity state variables in the 3D Mountain Car.

In Table 2 we focus on the best state variable mapping and show the MSE for each of the different possible action mappings. With the exception of Neutral, each task action has two source task actions with very similar error. This effect is caused by the doubling of state variables and actions when using 2D Mountain Car data as input to a 3D Mountain Car model. When using the state variable mapping described above, $\dot{x}_s$ is mapped to both $\dot{x}_t$ and $\dot{y}_t$. Consider saved source task data for the action Right. Right in the source task will cause $\dot{x}_s$ to increase. East in the target task will likewise cause $\dot{x}_t$ to increase, but it will not affect $\dot{y}_t$. Because $\dot{x}_s$ has been mapped to both of these state variables, one will be modified as the target task model expects for the action East, but the other will not. Intuitively, an appropriate action mapping would map both Right and Neutral from the source task to the action East in the target task. Because there is no clear single best 1-1 mapping, we choose to weight the different action mappings by the inverse of their measured MSE. Such a method will allow us to map multiple actions from the source task into the target task, weighted by their relative errors on our model.

Once the agent learns the mappings δ_X and δ_A (one-to-many for the state variable mapping and many-to-many for the actions), we construct the inter-task mappings χ_X and χ_A by taking the inverse of these mappings. We then use a transfer method which is very similar to that of *Q-Value Reuse* [20] (see Method 2). In this transfer method, the agent saves the 2D CMAC after training on the source task. In the target task, the agent modifies the

State Variable Mappings Evaluated

x	y	$\dot{x}$	$\dot{y}$	Avg. MSE	Best MSE
x	x	x	x	0.0384	0.0348
x	x	x	$\dot{x}$	0.0246	0.0228
x	x	$\dot{x}$	x	0.0246	0.0227
$\boldsymbol{x}$	$\boldsymbol{x}$	$\boldsymbol{\dot{x}}$	$\boldsymbol{\dot{x}}$	**0.0107**	**0.0090**
x	$\dot{x}$	x	x	0.0451	0.0406
x	$\dot{x}$	x	$\dot{x}$	0.0385	0.0350
x	$\dot{x}$	$\dot{x}$	x	0.0312	0.0289
x	$\dot{x}$	$\dot{x}$	$\dot{x}$	0.0245	0.0225
$\dot{x}$	x	x	x	0.0451	0.0406
$\dot{x}$	x	x	$\dot{x}$	0.0312	0.0290
$\dot{x}$	x	$\dot{x}$	x	0.0384	0.0350
$\dot{x}$	x	$\dot{x}$	$\dot{x}$	0.0245	0.0226
$\dot{x}$	$\dot{x}$	x	x	0.0516	0.0463
$\dot{x}$	$\dot{x}$	x	$\dot{x}$	0.0450	0.0407
$\dot{x}$	$\dot{x}$	$\dot{x}$	x	0.0450	0.0407
$\dot{x}$	$\dot{x}$	$\dot{x}$	$\dot{x}$	0.0383	0.0350

Table 1: This table shows the resulting MSE when using different state variable mappings. Each row shows a different mapping where the source task variables in the row are mapped to the target task variables at the head of the column (i.e., x, x, $\dot{x}$, $\dot{x}$ maps variable x_s to x_t, x_s to y_t, $\dot{x}_s$ to $\dot{x}_t$, and $\dot{x}_s$ to $\dot{y}_t$). The Avg. MSE column shows the MSE averaged over all possible action mappings for each row's state variable mapping. The Best MSE column shows the MSE for each row's state variable mapping when using the action mapping with the lowest MSE. Both metrics show that the state variable mapping in bold is significantly better than all other possible state variable mappings.

Action Mappings Evaluated

Target Task Action	Source Task Action	MSE
Neutral	Left	0.0118
Neutral	Neutral	0.0079
Neutral	Right	0.0103
West	Left	0.0095
West	Neutral	0.0088
West	Right	0.0127
East	Left	0.0144
East	Neutral	0.0095
East	Right	0.0089
South	Left	0.0099
South	Neutral	0.0093
South	Right	0.0135
North	Left	0.0136
North	Neutral	0.0100
North	Right	0.0100

Table 2: This table shows the MSE found when a source task action is mapped into a target task action. All experiments in this table use the same state variable mapping.

weights in a 4D CMAC when learning. However, when computing the action-value for a s, a pair, the agent also uses the saved 2D CMAC to evaluate the current position. Conceptually, $Q(s_t, a_t) = Q_{4DCMAC}(s_t, a_t) + Q_{2DCMAC}(\chi_X(s_t), \chi_A(s_a))$. However, as mentioned above, our action mapping is not one-to-one. Thus we iterate over all source task actions, multiply each by the inverse of the action mapping's recorded MSE, and then renormalize (see Method 2). The action mappings with the lowest error have the most influence on the value contributed by the source task CMAC. While learning, the target task CMAC's weights are modified by Sarsa(λ) and will allow for an accurate approximation of the action-value function, even though the transferred source CMAC (which remains unchanged) will not be optimal in the target task.

Method 2 Q-Value Reuse in 3D Mountain Car

$x, y, \dot{x}, \dot{y} \leftarrow$ agent's current state
$a_t \leftarrow$ action to evaluate
for each source task action a_s **do**
    SUM += $1/MSE_{a_t,a_s}$
for each source task action a_s **do**
    $Q(s, a_t)$ += $Q_{2dCMAC}(x, \dot{x}, a_s) \times 1/\text{SUM} \times 1/(\text{MSE}_{a_t,a_s})$
    $Q(s, a_t)$ += $Q_{2dCMAC}(y, \dot{y}, a_s) \times 1/\text{SUM} \times 1/(\text{MSE}_{a_t,a_s})$
$Q(s, a_t)$ += $Q_{4dCMAC}(x, y, \dot{x}, \dot{y}, a_t)$

5.2 Transfer from 2D to 3D Mountain Car

Figure 3 shows learning curves in 3D Mountain car, each averaged over 25 independent trails. For each trial, after each episode we evaluate the policy off-line without exploration. To graph the learning curve we average all 25 learning curves for the previous 10 episodes and plot the mean. First, consider the lines "Without transfer" and "Average Both." Average Both transfers by averaging over all action mappings and all state variable mappings. Such a method can be considered a type of blind transfer – no time or samples are spent learning an inter-task mapping, but the resulting learning curve is much worse than learning without transfer. Evidently, transferring without any consideration to the state and action variable mappings may be quite harmful to learning. However, as is shown by the other transfer experiments, using MASTER to learn these mappings can enable transfer that is quite beneficial.

The line "Transfer: 1/MSE" is generated by transferring from 100 episodes of 2D Mountain Car where the action mapping is weighted by the inverse of its observed MSE in the target task model. Using paired t-tests we find that the 1/MSE transfer curve is statistically significantly better, at the 95% level, than learning without transfer for episodes 2–473.[4] Also included in the graph are three other transfer mappings for comparison. "Hand Coded" uses hand-coded state variable and action mappings based on our knowledge of the domains as humans. We believe that this learning curve represents the upper bound on transfer for 100 episodes of 2D Mountain Car. It is encouraging that the 1/MSE learning curve quickly converges to the same asymptotic value as the hand coded transfer learner. "Average Actions" performs transfer with the learned state variable mapping but simply averages over all actions. This would be equivalent to all the possible action mappings having the same error, and indicates how important using an action mapping is for efficient transfer. Figure 4 shows a magnified version of the graph to better see differences between the different transfer methods.

We also tested a final method for weighting the different action mappings. Rather than using all action mappings and weighting by the inverse of the MSE, we selected only the best or two best actions. The learning curve resulting from this method was qualitatively similar to the 1/MSE learning curve and is not shown.

5.3 Reducing the Total Sample Complexity

The results in Figure 3 show that learned source task knowledge can be effectively used with a learned mapping. Thus, if an agent has already trained on 2D Mountain Car and wants to learn 3D Mountain Car, it likely makes sense to use its past knowledge rather than to learn without it. However, consider a situation where the agent has not trained on 2D Mountain Car and is faced with the 3D Mountain Car task. Should it first train on the 2D task, learn a mapping, and then transfer? Or should it directly tackle the more difficult 3D task?

To help answer this question, we varied the amount of data used in the source and target task to learn a mapping, as well as how many episodes in the source task used to learn the 2D CMAC's weights. Earlier experiments showed transfer after learning for 100 episodes in the source task, spending 50 episodes collecting data

[4]On the first episode, the agent with transferred knowledge has an average reward of -4640 while the agent learning without transfer has an average reward of -5000, which is not different at the 95% confidence level.

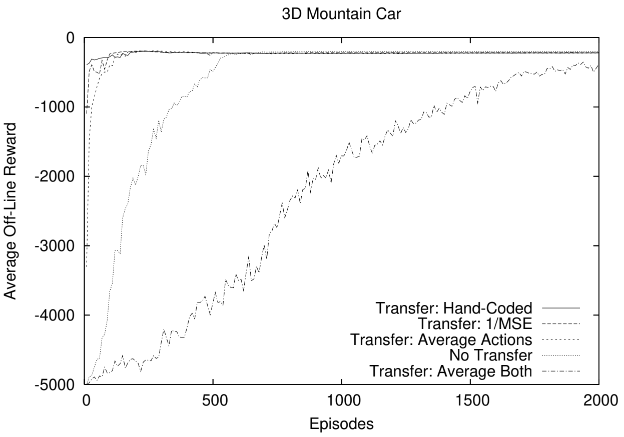

Figure 3: This graph compares learning without transfer to: transfer with learned state variable and action mappings, transfer with hand-coded mappings, transfer with mappings that average over all possible mappings, and transfer with a learned state variable mapping. Figure 4 zooms in on the beginning of the same curves. Each learning curve averages 25 independent trials.

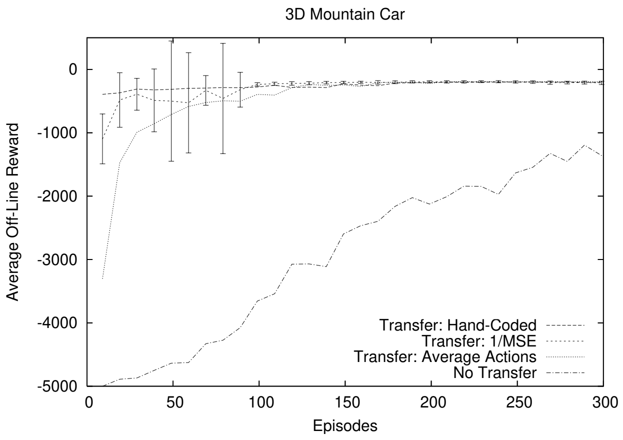

Figure 4: This graph shows the same curves as in Figure 3. The hand-coded mapping performs slightly better than the fully learned mapping, which in turn is better than using only the state variable mapping. To help visualize the magnitude of the evaluation noise, the learned mapping transfer curve shows error bars at $\pm$ 1 standard deviation. Each learning curve averages 25 independent trials.

in the target task, and then using MASTER with transfer to learn the target task. We tried using 100, 50, 25, and 10 episodes of source task training, as well as 50, 25, and 10 episodes of target task training. We found that only when we reduce the number of source task episodes to 10 does performance degrade.

Figure 5 compares learning 3D Mountain Car without transfer to using transfer. The agent trains for 25 episodes in the source task, collects data for 10 episodes in the target task, uses MASTER to learn the inter-task mappings, weights the action mappings by 1/MSE, and then learns in the target task. Note that the transfer learning curve has been shifted by 35 episodes (the first graphed point is at episode 45, instead of at episode 10) to explicitly account for the episodes spent before learning in the target task. A series of paired t-tests show that the difference between learning without transfer and learning with transfer while accounting for all episodes used is statistically significantly different at the 95% level from learning without transfer from episodes 36–474. We therefore conclude that for some tasks, it may be in an agent's interest to train first on a simple source task, learn a mapping, and then learn on a target task, rather than learn on the target task directly. [5]

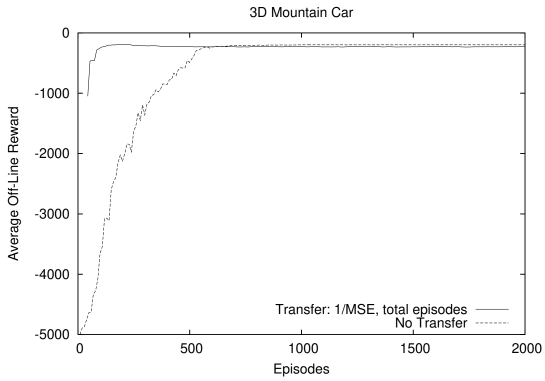

Figure 5: This graph compares learning without transfer to transfer using learned mappings. The transfer learning curve does not start at 0 episodes as it now reflects the total number of episodes used to learn the mappings in the source and target task. This result shows that the total time to learn a source task, an inter-task mapping, and then learn in a target task may less than learning a target task directly. Each learning curve averages 25 independent trials.

5.4 Comparison to Previous Work

We would like to compare our method with previous methods for learning a mapping using data from the environments (i.e. [15, 18, 21]) but the 2D and 3D Mountain Car tasks do not easily subdivide into groups of state variables. For instance, our previous work [21] presents an example from a logistic domain that divides the world into two object types, trucks and locations, and supposes that the source and target tasks can have different numbers of these objects. However, in Mountain Car there is no clear division of "object types." To enable a comparison, we will decide to group state variables into (position, velocity) tuples. Our source task will thus have one object, $(x, \dot{x})$, and the target task will have two objects, $(x, \dot{x})$ and $(y, \dot{y})$. Note that a significant amount of information about the relationship between the two tasks has already been encoded in this formulation.

We follow the procedure of Taylor, Whiteson, and Stone. In the source task, we collect experience while learning in the form $(x_s, \dot{x}_s,\ a_s, r,\ x'_s, \dot{x}'_s)$, where the s subscript denotes the source task. After learning, we use the data to train an action classifier: $C_{action}(x_s, \dot{x_s}, r, x'_s, \dot{x_s}') \ \mapsto\ a_s$. Then, in the target task, we collect data in the form $(x_t, \dot{x}_t,\ y_t, \dot{y}_t, a_t, r,\ x'_t, \dot{x}'_t,\ y'_t, \dot{y_t}')$. After collecting the target task data, we use the action classifier to predict which similar source task action was used for an observed target task tuple. For instance, the output from the action classifier $C_{action}(x_t, \dot{x_t}, r, x'_t, \dot{x_t}')$ would give some source task action. The returned source task action is counted as a vote that the target task action associated with this tuple, a_t, is the same as the action returned by the classifier, a_s. Note that no state variable classifier is needed, as there is only one object type. Thus, $(x_t, \dot{x}_t)$ and $(y_t, \dot{y}_t)$ both get mapped to $(x_s, \dot{x}_s)$ because of the knowledge we implicitly gave the agent in how we chose the state variable grouping. χ_X has been provided by human knowledge, but the classifier is responsible for learning χ_A.

[5]The learning parameters for the 3D Mountain Car task were tuned for learning without transfer. In a different series of experiments, not shown, the transfer learning curves were improved by re-tuning the learning parameters.

We collected 50 episodes of data in 2D Mountain Car and trained a neural network action classifier with 5 inputs (four state variables and the current reward) to predict the source task action that was taken. The neural network was unable to learn to correctly classify the data until we changed the agent's policy so that the car took each action for 5 successive time steps. By grouping successive states together (i.e., instead of using the state at times t and t+1, we used the state at times t and t+5), the effects of actions outweighed the effects of gravity and we were able to learn to accurately classify source task actions. The action mapping learned is similar to the results of our method, as expected (see Table 3). If we use this action mapping to learn in 3D Mountain Car (not shown), weighting the different actions by the number of "votes" each mapping received, we find that the target task learning is very similar to our 1/MSE method using both the learned state variable and action mappings described above. The main significance of this result is that it confirms that MASTER is able to find an action mapping similar to that found by an existing learning method, even though significantly less human knowledge is required.

Action Mappings via Classification

	Left	Neutral	Right
Neutral	679	18910	154
West	8518	10554	184
East	285	10046	9177
South	8773	10730	186
North	375	10093	9540

Table 3: This table shows the confusion matrix when evaluating 3D Mountain Car data on an action classifier trained using 2D Mountain Car data. Each value in the matrix is the number of times a target action (row) was classified as a source action (column), and each data can be considered a vote for the action mapping.

5.5 Transfer in Hand Brake Mountain Car

In this section we examine transfer into the 3D Hand Brake Mountain Car task. First, consider an agent that has previously trained for 500 episodes of 2D Hand Brake Mountain Car. Figure 6 compares learning without transfer, learning after transferring only the state variable mapping, and learning after transferring both the state variable and action mapping. This result confirms that MASTER can learn a useful inter-task mapping in this variant of Mountain Car.

Consider an agent that has previously trained on 2D Mountain Car, both with and without a hand brake action. If the agent is now tasked with 3D Hand Brake Mountain Car, it should be able to learn mappings for both tasks and use the learned mappings to intelligently transfer from the source tasks. One option would be to select the source task with learned mappings that had the lowest MSE, which in this case would be 2D Mountain Car with a hand brake action (see Table 4 for a partial summary). A second option would be to weight the mappings from both tasks by the inverse of their recorded MSEs. Figure 7 shows both of these methods outperform transferring only from the 2D Mountain Car without hand brake, as well as outperforming learning the 3D hand brake task without transfer. Interestingly, transferring from both source task appears better than transferring from a single source task (although the differences are not statistically significant at the 95% level due to high variance).

This experiment shows that it is possible to leverage MASTER's evaluation of different inter-task mappings to help determine how similar tasks are. It is possible that such a method could also be used to learn when an action or state variable in the target has no analog in any source task, but we leave this enhancement to future work. In our experiment, the transition function of the two hand brake tasks was more similar than the 3D hand brake task and the

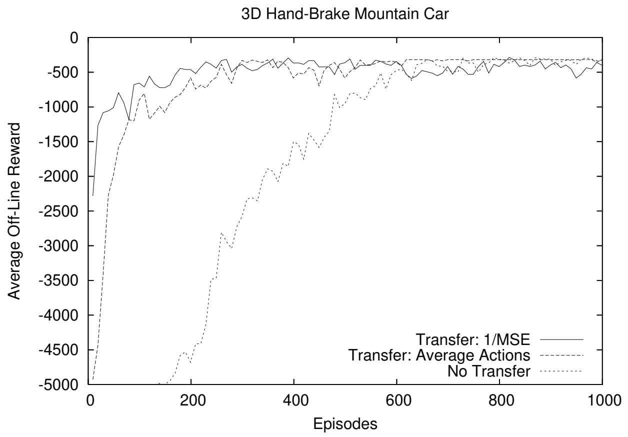

Figure 6: This graph shows that transfer using both learned mappings outperforms both learning without transfer and using only the learned state variable mappings. Each learning curve averages 25 independent trials.

Action Mappings for 2 Source Tasks

Target Task Action	Source Task Action	MSE for 2D as Source Task	MSE for 2D Hand Brake as Source Task
Neutral	Left	0.0196	0.0140
Neutral	Neutral	0.0188	0.0113
Neutral	Right	0.0244	0.0162
Neutral	Hand Brake		0.0665
West	Left	0.0180	0.0111
West	Neutral	0.0226	0.0143
West	Right	0.0320	0.0219
West	Hand Brake		0.0678
		...	
Hand Brake	Left	0.1673	0.1284
Hand Brake	Neutral	0.1706	0.1285
Hand Brake	Right	0.1985	0.1360
Hand Brake	Hand Brake		0.0097

Table 4: This table shows some of the MSEs found when a source task actions from 2D Mountain Car (with and without a hand brake action) are mapped into a a 3D Hand Brake Mountain Car action. Note that the errors for the 2D hand brake task are less than the standard 2D task and that no source task action from the standard 2D task maps well to the 3D hand brake action.

2D non-hand brake task. While encouraging, such a metric only accounts for the similarity of two tasks' transition functions. If, for instance, the target task's goal state were moved from $(0.5, 0.5)$ to $(-1.2, -1.2)$, it is unlikely that transferring from either 2D Mountain Car task would improve learning. In fact, when transferring from such mismatched tasks, it is possible that transfer would *hurt* the learner's performance, relative to learning without transfer. Insulating an agent from the effects of such *negative transfer* is a difficult problem that we leave to future work, along with refining this proposed task similarity metric to account for differences in source and target tasks' reward functions.

6. FUTURE WORK

In this work we have focused on reducing the sample complexity of learning by showing that MASTER can increase performance in a target task with effective reuse of past knowledge, as well as showing that the total number of episodes can be effectively reduced with an automatically learned mapping. We do so under the assumption that for many fielded agents, sample complexity is much more of a bottleneck than computational complexity. In the future we would also like to examine reducing computational complexity.

The first area for improvement would be tackling the inner loop of MASTER which is exponential in the number of state variables

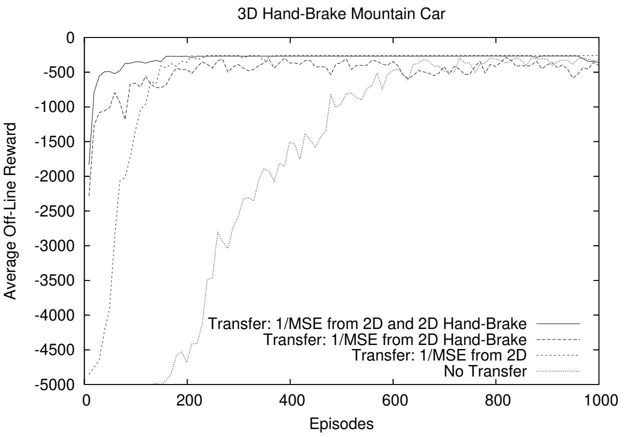

Figure 7: This graph compares learning the 3D Hand Brake Mountain Car task without transfer, with transfer from the 2D Hand Brake Mountain Car, with transfer from the 2D Mountain Car, and with transfer from both versions of the 2D task (weighted by the inverse of their respective mapping errors). Each learning curve averages 25 independent trials.

and actions. As suggested before, if this method is to scale to tasks with hundreds of state variables or actions, some sort of heuristic search would be needed, rather than an enumeration of all possible mappings. However, we reiterate that the main insight of MASTER is that the different possible mappings can be evaluated off-line, and that utilizing more powerful search techniques for discovering an optimal mapping is left to future work.

It would also be useful to determine if the sample complexity could be further reduced. One idea would be to explore in the target task so as to minimize the uncertainty in the target task transition model, making exploration more efficient. Another possible tact would be to interleave building the target task transition model and gathering data in the target task. By examining the learned model, it may be possible to continue exploring in the target task only as long as collected data is changing the model significantly.

MASTER relies on being able to explore in the target task quickly and build an approximate model. However, in some tasks the initial exploration may not be indicative of the entire MDP and a model learned with only a little training data would be misleading. While there is likely no way to guard against this for arbitrary MDPs, it would be useful to be able to define the type of task for which such initial exploration is likely to yield a useful model for learning a mapping.

Lastly, we note that this work focuses on pairs of tasks drawn from the same domain. While other work has demonstrated that *cross-domain* transfer is possible [19], it is likely that autonomously learning mappings will become more difficult when the source and target tasks become less similar.

7. CONCLUSION

This paper has introduced MASTER, a method for automatically learning a mapping between tasks. We have empirically demonstrated the efficacy of this algorithm on a series of tasks in the Mountain Car domain. These results show that a learned task mappings can effectively increase the speed of learning in a novel target task so that the sample complexity is reduced using transfer, relative to learning without transfer. Additionally, we show an initial approach for leveraging learned inter-task mappings to assist with the problem of appropriate source task selection.

Acknowledgments

We would like to thank Lilyana Mihalkova 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.

8. REFERENCES

[1] A. Agnar and P. Enric. Case-based reasoning: Foundational issues, methodological variations, and system approaches, 1994.
[2] J. S. Albus. *Brains, Behavior, and Robotics*. Byte Books, Peterborough, NH, 1981.
[3] S. Dzeroski, L. D. Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):5–52, April 2001.
[4] F. Fernandez and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proc. of the 5th International Conf. on Autonomous Agents and Multiagent Systems*, 2006.
[5] M. Genesereth and N. Love. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2), 2005.
[6] G. Kuhlmann and P. Stone. Graph-based domain mapping for transfer learning in general games. In *Proceedings of The Eighteenth European Conference on Machine Learning*, September 2007.
[7] 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.
[8] R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.
[9] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178, 2002.
[10] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7:155–162, 1964.
[11] 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.
[12] G. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University, 1994.
[13] M. Sharma, M. Holmes, J. C. Santamaria, A. Irani, C. Isbell, , and A. Ram. Transfer learning in real-time strategy games using hybrid cbr/rl. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.
[14] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.
[15] 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.
[16] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.
[17] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.
[18] E. Talvitie and S. Singh. An experts algorithm for transfer learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.
[19] M. E. Taylor and P. Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, June 2007.
[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 AAMAS Conf.*, May 2007.
[22] L. Torrey, T. Walker, J. W. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *The 16th European Conf. on Machine Learning*, 2005.
[23] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.