An Autonomous Transfer Learning Algorithm for TD-Learners

Anestis Fachantidis[1], Ioannis Partalas[2], Matthew E. Taylor[3], and Ioannis Vlahavas[1]

[1] Department of Informatics, Aristotle University of Thessaloniki
{afa,vlahavas}@csd.auth.gr
[2] Laboratoire LIG, Université Joseph Fourier
ioannis.partalas@imag.fr
[3] School of Electrical Engineering and Computer Science, Washington State University
taylorm@eecs.wsu.edu

Abstract. The main objective of transfer learning is to use the knowledge acquired from a source task in order to boost the learning procedure in a target task. Transfer learning comprises a suitable solution for reinforcement learning algorithms, which often require a considerable amount of training time, especially when dealing with complex tasks. This work proposes an autonomous method for transfer learning in reinforcement learning agents. The proposed method is empirically evaluated in the keepaway and the mountain car domains. The results demonstrate that the proposed method can improve the learning procedure in the target task.

1 Introduction

In recent years, a wealth of *transfer learning* (TL) methods have been developed in the context of *reinforcement learning* (RL) tasks. Typically, when an RL agent leverages TL, it uses knowledge acquired in one or more (*source*) tasks to speed up its learning in a more complex (*target*) task[1].

Although the majority of the work in this field presumes that the source task is connected in an obvious or natural way with the target task, this may not the case in many real life applications where RL transfer could be used. These tasks may have different state and action spaces or even different reward and transition functions. One way to tackle this problem is to use functions that map the state and action variables of the source task to state and action variables of the target task. These functions are called *inter-task mappings* [1]

While inter-task mappings have indeed been used successfully in several settings, we identify two shortcomings. First, an agent typically uses a hand-coded mapping, requiring the knowledge of a domain expert. If human intuition cannot be applied to the problem (e.g. transferring knowledge for robotic joint control between robots with different degrees of freedom), selecting an inter-task mapping may be done randomly requiring extensive, costly experimentation and time not typically available in complex domains or in real applications. Second, even if a correct mapping is used, it is fixed and applied to the entire state-action space, ignoring the important possibility that different mappings may be better for different regions of the target task.

In this paper we propose a generic method for the automated on-line selection of inter-task mappings in transfer learning procedures. The key insight of the proposed method is to select the inter-task mapping for a specific state-action of the target task based on the corresponding source task state-action values.

The main contributions of this work are to 1) Present a novel method for value-based mapping selection, providing its theoretical foundations and an empirical comparison of two different mapping selection criteria; 2) Alleviate the problem of predefining a mapping between a source and target task in TL procedures by presenting a fully automated method for selecting inter-task mappings; 3) Introduce and evaluate a novel algorithm that implements the proposed method.

Experimental results demonstrate success of the proposed algorithm in two RL domains, Mountain Car and Keepaway. Some part of the Keepaway results presented here has been presented also in a workshop [2]. The extensive results and the analysis in this work support the method's key insight on the appropriate mapping selection criteria for value-based mapping selection methods.

2 Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) addresses the problem of how an agent can learn a behaviour through trial-and-error interactions with a dynamic environment [3]. In an RL task the agent, at each time step t, senses the environment's state, $s_t \in S$, where S is the finite set of possible states, and selects an action $a_t \in A(s_t)$ to execute, where $A(s_t)$ is the finite set of possible actions in state s_t. The agent receives a reward, $r_{t+1} \in \Re$, and moves to a new state s_{t+1}. The general goal of the agent is to maximize the expected return, where the return, R_t, is defined as some specific function of the reward sequence.

2.2 Transfer Learning

Transfer Learning refers to the process of using knowledge that has been acquired in a previously learned task, the *source task*, in order to enhance the learning procedure in a new and more complex task, the *target task*. The more similar these two tasks are, the easier it is to transfer knowledge between them. By similarity, we mean the similarity of their underlying Markov Decision Processes (MDP) that is, the transition and reward functions of the two tasks as also their state and action spaces.

The type of knowledge that can be transferred between tasks varies among different TL methods. It can be value functions, entire policies as also a set of samples from a source task used from a batch RL algorithm in a target task [1]. In order to enable transfer learning across tasks with different state variables (i.e., different representations of state) and action sets, one must define how these tasks are related to each other. One way to represent this relation is to use a pair $\langle X^S, X^A \rangle$ of inter-task mappings [1], where the function $X^S(s_i)$ maps one target state variable to one source state variable and $X^A(a)$ maps an action in the target task to an action in the source task (see Table 1 for an example of states mapping).

3 Multiple Inter-Task Mappings in TD Learners

This paper proposes a value-based transfer method which is also able to autonomously select the appropriate inter-task mapping. As this transfer method applies to TD-learners it doesn't require any information about the environment, such as a model of its transition or reward functions, therefore being a more computationally efficient method capable for on-line selection of the appropriate mappings.

We assume that an RL agent has been trained in the source task and that it has access to a function Q_{source}, which returns an estimation of the Q value for a state-action pair of the source task. The agent is currently being trained in the target task, learning a function Q_{target} and senses the state s_τ. In order to transfer the knowledge from the source task, the method selects the best state-action mapping $X_{best} = \langle X^S_{best}, X^A_{best} \rangle$, under a novel criterion described next in this text, and adds the corresponding values $Q_{source}(X^S(s_\tau)), X^A(a_\tau))$ from the source task via the selected mapping to the target task values $Q_{target}(s_\tau, a_\tau)$. In our proposed method the best mapping is defined as the mapping that returns the state-action values that had the maximum mean value in the source task compared to the state-action values returned by the other mappings, that is

$$X_{best} = \arg\max_{X} \sum_{a \in A_{source}} \frac{Q_{source}(X^S(s_\tau), a)}{|A_{source}|}$$

There are alternate ways to select the best mapping based only on the Q-values of the source task, such as selecting the mapping with the maximum action value, but such a choice would imply that a source task action is correctly mapped beforehand and also that the best action for the source task policy is also the best for the target task policy. In order to avoid such assumptions, we use instead the maximum mean Q-Value as our transfer heuristic. Moreover, we analytically argue on the natural meaning and correctness of such a choice.

Specifically, as our value-based transfer method belongs to a family of methods such as Q-value reuse [1], we note two things. Every transfer method that shapes a target task Q-function using a source task's Q-values, is based on two main assumptions:

- The most meaningful and correct inter-task mapping is known beforehand and is used to map the state and action variables of the target task's Q-function to the source task's Q-function.
- Using this inter-task mapping, a greedy ($\max Q$-value) target task policy derived exclusively from the (mapped) source task Q-function, is meaningful in the target task and assists towards its goal.

Simply said, the second assumption implies that given a correct mapping, a high value state-action pair in the source task is expected to be of high value in the target task too. Since our proposed method belongs to this family of methods but with the major difference that we don't manually set (and presume) the best mapping, but instead aim to find it, the first assumption is not met and should at least be relaxed.

Concretely, not necessarily meeting the first assumption means that a target task policy derived by greedily ($\max Q(s, a)$) using the source task Q-function is also not necessarily meaningful. This means that selecting mappings for the current target task

state based on the maximum source task Q-value implies not only that the actions are correctly mapped but that they are also the best for the target task policy.

To relax these assumptions we don't presume the immediate and greedy next-step use of the source task policy (implied when one uses the maximum $Q^{\pi}(s,a)$ for the agent's next step) and instead we use the value $V^{\pi}(s)$ of that source task state but under a different source task policy, the *random next-step* policy. In this policy the agent chooses a random action for the current state and then uses π for the next step and after that.

The random next-step policy protects us from the violation of the assumptions stated above, if we would greedily use the source task Q-function ($\max Q$-value). It is actually **a parsing policy of the source task's state-action values**. Considering this policy leads us to transferring based on the mean Q-value returned by each mapping for the agent's next step.

Specifically, consider the state value function $V^{\pi}_{source}(s)$ in the source task. Although known from basic MDP theory, we analytically calculate the following in order to demonstrate the natural meaning of the mean Q-value choice:

$$V^{\pi}_{source}(s) = E_{\pi}\{R_t | s_t = s\}$$

Based on the conditional expectation theorem and conditioning on all possible actions, the above is equal to:

$$= \sum_{\alpha} E_{\pi}\{R_t | s_t = s, \alpha_t = \alpha\} P\{\alpha_t = \alpha | s_t = s\}$$

The probability term above is the policy $\pi(s,\alpha)$, so:

$$= \sum_{\alpha} E_{\pi}\{R_t | s_t = s, \alpha_t = \alpha\} \pi(s,\alpha)$$

Since the expectation term above, is $Q^{\pi}(s,\alpha)$ the above is equal to:

$$= \sum_{\alpha} Q^{\pi}(s,\alpha)\pi(s,\alpha)$$

Since under our policy the next action in a state s is chosen randomly (uniformly): $\pi(s,\alpha) = \frac{1}{|A_{source}(s)|} = \frac{1}{k}$ so

$$V^{\pi}_{source}(s) = \frac{1}{k} \sum_{\alpha} Q^{\pi}(s,\alpha) \tag{1}$$

And so using a random policy just for the next step and then assuming following π as usual, implies that the value of the state s when following our modified policy, equals the mean Q-value of the state s (RHS of Equation 1). As we mentioned above using the next-step random policy protects our proposed method from an "assumption over assumption" pitfall.

The above insight is implemented in our proposed TL algorithm, Value-Addition. Algorithm 1 shows the pseudo-code of the transferring procedure.

Algorithm 1 Value-Addition procedure: Multiple Mappings in TD Learners

```
1:  procedure VALUE-ADDITION(s_τ, X, Q_target, Q_source)
2:      bestMeanQValue ← 0; bestMapping ← 0
3:      N ← |X|                                   ▷ the number of mappings considered
4:      for i ← 1 ... N do
5:          s ← X_i^S(s_τ)
6:          meanQValue ← 0
7:          for all a ∈ A_source(s) do
8:              meanQValue ← meanQValue + Q_source(s, a)
9:          meanQValue ← meanQValue/|A_source(s)|
10:         if meanQValue > bestMeanQValue then
11:             bestMeanQValue ← meanQValue
12:             bestMapping ← i
13:     χ_S ← X^S_bestMapping
14:     χ_A ← X^A_bestMapping
15:     s ← χ_s(s_τ)
16:     for a ∈ A_source(s) do
17:         Q_target(s_τ, χ_A^{-1}(a)) ← Q_target(s_τ, χ_A^{-1}(a)) + Q_source(s, a)
```

On lines 3–12, the algorithm finds the best mapping for the current target task state s_τ from instances that are recognized in the target task.

After the best mapping is found, the algorithm adds the Q-values from the source task to the Q-values from the target task (lines 13–16). Through the inverse use of the best action mapping, χ_A^{-1} the algorithm updates the value of the equivalent target task action. Note that if a target action is not mapped to a source action, the algorithm does not add an extra value. Finally, the updated Q-values in the target task can be used for a regular TD update, action selection, etc.

4 Domains

4.1 Keepaway

Keepaway [4], is a subset of the RoboCup robot soccer domain, where K keepers try to hold the ball for as long as possible, while T takers (usually $T = K - 1$) try to intercept the ball (Figure 1). The agents are placed within a fixed region at the start of each episode, which ends when the ball leaves this region or the takers intercept it.[4]

The task is modelled as a semi-Markov decision process (SMDP), as it defines macro-actions that may last for several time steps. The available macro-actions for a keeper with ball possession are *HoldBall* and *Pass-k-ThenReceive*, where k is another keeper. A keeper executing *HoldBall* remains stationary. A keeper executing *Pass-k-ThenReceive* performs a series of actions in order to pass the ball to team-mate k and then executes the *Receive* sub-policy: If no keeper possesses the ball and he is the closest keeper to the ball then he executes macro-action *GoToBall*, otherwise he executes

[4] For more information please refer to the original paper [4].

macro-action *GetOpen* in order to move to an open area. *Receive* is also executed by keepers when they don't possess the ball.

The features that describe the state of the environment for a keeper K_1 that possesses the ball are: a) the distances of all agents to the center of the field, b) the distance of all other players to K_1, c) the distances of K_1's team-mates to the closest opponent, and d) the minimal angles between K_1's team-mates and an opponent with the vertex at K_1.

The task becomes harder as extra keepers and takers are added to the fixed-sized field, due to the increased number of state variables on one hand, and the increased probability of ball interception in an increasingly crowded region on the other.

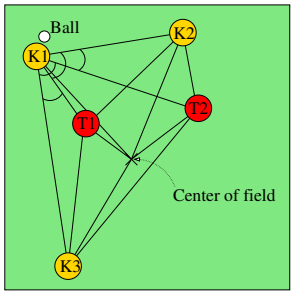

Fig. 1. 3 vs. 2 Keepaway

4.2 Mountain Car

Our experiments use the mountain car domain [5]. The standard 2D task (MC2D) requires an under-powered car to drive up a hill. The state is described by two continuous variables, the position $x \in [-1.2, 0.6]$, and velocity $v_x \in [-0.07, 0.07]$. The actions are $\{$Neutral, Left, Right$\}$, and the goal state is $x >= 0.5$, with a reward of -1 on each time step. The 3D mountain car (MC3D) task extends the 2D task by adding an extra spatial dimension [6]. The state is composed by four continuous variables. The coordinates x and y are in $[-1.2, 0.6]$, and the velocities v_x and v_y are in $[-0.07.0.07]$. The actions are $\{$Neutral, West, East, South, North$\}$, the goal is $x, y >= 0.5$ and the reward is -1 on each time step.

5 Experiments

5.1 Transferring with Value-Addition in Mountain Car 3D

First, an agent is trained in the Mountain Car 2D task (source task). The algorithm that is used for training in the source task as also in the target task, is the Sarsa(λ) using linear tile-coding (CMAC) with 14 tilings. The learning rate α is set to 0.5. The ϵ parameter is set to 0.1, which is multiplied by 0.99 at each episode, and λ to 0.95. These settings are the default in the Mountain Car package and they are the best found so far. While acting in the source task, the agent is recording the weights of its CMAC function approximator for each tiling and action. The resulting data file captures the state of the source task function approximator and is the knowledge we need to transfer and use in the target task. Finally, in the source task the agent was trained for 1600 episodes.

For the MC3D task we use the same settings as above except for α which is set to 0.2. We select this setting as it was found to yield very good results [7]. For both the MC2D and MC3D tasks we set the maximum steps of an episode to 5000. If the agent exceeds the maximum number of steps without reaching the goal state, then the episode ends and the agent is placed at the bottom of the hill.

3D variable	2D variable	3D action	2D action
$\mathbf{x}$	$\mathbf{x}$	Neutral	Neutral
$\mathbf{y}$	-	West	Left
$\mathbf{v_x}$	$\mathbf{v_x}$	East	Right
$\mathbf{v_y}$	-	South	-
		North	-

Table 1. An intuitive inter-task mapping in Mountain Car mapping correctly same-type state variables and similar effect action variables.

In this experiment, Value-Addition uses a pool of 7 mappings that we manually set before the experiment's execution. The first four mappings we set are considered intuitive since they map position state variables of the MC2D to position state variables in MC3D and consequently the velocity state variables of MC3D to the velocity state variable of MC2D. See Table 1 for an example of an intuitive mapping. For the other three mappings, the first one maps in an intuitively correct way the action variables but not the state variables. The second one maps the states intuitively but maps actions in a different way (as an example, it maps the action Neutral of MC2D to the action East in MC3D) Finally the third mapping maps both state and action variables in a non-intuitive way.

For comparison, various other algorithm were tested besides Value-Addition. Each algorithm was selected in order to examine some specific hypothesis of this study: i) A standard Sarsa(λ) agent without transfer, in order to evaluate the quality and feasibility of transfer using Value-Addition; ii) an agent implementing transfer learning using random mappings from the pool of the 7 mappings described earlier, in order to demonstrate the importance of using the correct mapping; iii) an agent using the single most intuitive mapping with the Q-Value Reuse algorithm [1], in order to demonstrate the ability of Value addition to achieve similar performance to a transfer algorithm that has been given beforehand a correct mapping and is not able to autonomously change that selection - and finally iv) an agent using Value-Addition but transferring from the mapping that has the maximum action value (see Section 3) in order to evaluate the proposed criterion of using instead, the mean state-action value.

Figure 2 depicts the results of this experiment using the time-to-threshold metric. The x axis shows various thresholds representing the steps needed by the agent to find the goal (exit) and the y axis represents the corresponding training time needed to achieve these goals. Each curve averages the results from 15 runs of it's corresponding algorithm.

The Value-Addition shows a statistically significant (at $p < 0.05$) performance increase from the no-transfer case. Using a random mapping implements negative transfer since it's performance is worse than the no-transfer case. Clearly this shows the importance of correctly selecting an inter-task mapping. Moreover, Value-Addition in its standard set-up performs better than value addition using the mapping that has the maximum action value (at $p < 0.05$). This finding confirms the method's key intuition described in section 3, for selecting a mapping based on its average state-action value and not on their maximal one. Even more interestingly value addition demonstrates better (not stat. sig.) performance than the single-mapping algorithm (Q-Value Reuse). This finding is important since as it is mentioned above, the single mapping algorithm

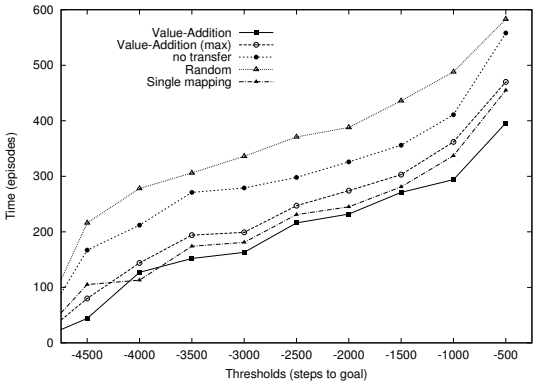

Fig. 2. Time-to-threshold metric for different thresholds in the 3D MC.

Mapping	Mapping Description	Percentage	mean x	mean y	mean vx	mean vy
0	intuitive	25.1	-0.427	-0.453	-0.003613	0.00346525
1	intuitive	34.7	-0.458	-0.141	0.00311873	-0.00271896
2	intuitive	0	-	-	-	-
3	intuitive	0	-	-	-	-
4	intuitive actions	40.2	-0.473	-0.451	0.001659	0.00245182
5	intuitive states	0	-	-	-	-
6	not intuitive	0	-	-	-	-
All		100	-0.456816	-0.344	0.000842	0.0009142

Table 2. The seven available mappings for Value-Addition in MC, the percentage of the total time and the average state in which they were selected.

was using an intuitively-correct mapping given to it beforehand whereas Value - Addition had to discover it.

Furthermore, we conducted an analysis on the mappings selected by Value-Addition in MC3D. Table 2 shows how many times Value-Addition used each of the 7 mappings (described earlier in this section). The four rightmost columns of the table show the mean state values of the states where each mapping was used. Also, the last row represents the general case independently from the mapping selected. We can observe how each mapping's mean state value differentiates from that of "all mappings" mean state value showing that Value-Addition selects mappings also in relation to the specific state the agent is. As an example, mapping #4 is used more in west and south positions of the state space because it's mean x and y values are smaller that the corresponding ones of "All".

Interestingly, the mappings analysis shows that the mappings selected were at a 59.8% those considered as intuitive and correct for MC . Moreover, Value-Addition may also select a mapping that maps state variables in a non intuitive way (i.e. a velocity state variable to a position state variable), but it never selects a non-intuitive action mapping (it finds the intuitive ones). The proposed method is more sensitive on the action mappings differences than that among state mappings. This behaviour is explained by the fact that different state mappings may have the same average action value but different action mappings tend to have different average action values thus resulting to a behaviour particularly able to differentiate and select among action mappings.

5.2 Transferring with Value-Addition in Keepaway

This section evaluates the proposed approach in Keepaway. The dimensions of the Keepaway region are set to 25m $\times$ 25m and remains fixed for all source and target tasks. The algorithm that is used to train the keepers is the SMDP variation of Sarsa(λ) [3]. Additionally, we use linear tile-coding for function approximation with settings shown to work well in the Keepaway domain [8].[5]

To evaluate the performance of the proposed approach in Keepaway we also use the time-to-threshold metric. In Keepaway this threshold corresponds to a number of seconds that keepers maintain ball possession. In order to conclude that the keepers have learned the task successfully, the average performance of 1,000 consecutive episodes must be greater than the threshold. We compare (1) the time-to-threshold without transfer learning with (2) the time-to-threshold with transfer learning **plus the training time in the source task**. Finally, an important aspect of the experiments concerns the way that the mappings are produced. For the Keepaway domain, the production of the mappings can be automatic. More specifically, any K^t vs. T^t task can be mapped to a K^s vs. T^s task, where $K^s < K^t$ and $T^s < T^t$, simply by deleting $K^t - K^s$ team-mates and $T^t - T^s$ opponents of the keeper with ball possession. The actual number of different mappings is:

$$|X| = \binom{K^t - 1}{K^s - 1}\binom{T^t}{T^s} = \frac{(K^t - 1)!T^t!}{(K^s - 1)!(K^t - K^s)!T^s!(T^t - T^s)!}$$

Transfer into 4 vs. 3 from 3 vs. 2 This subsection evaluates the performance of the proposed approach on the 4 vs. 3 target task using a threshold of 9 simulated seconds. We use the 3 vs. 2 task as the source and experiment with different number of training episodes, ranging from 0 (no transfer) to 3,200.

Table 3 shows the training time and average performance (in seconds) in the source task, as well as time-to-threshold and total time in the target task for different amount of training episodes in the source task. The results are averaged over 10 independent runs and the last column displays the standard deviation. The time-to-threshold without transfer learning is about 13.38 simulated hours.

We first notice that the proposed approach leads to lower time-to-threshold in the target task compared to the standard algorithm that does not use transfer learning. This is due to the fact that the more the training episodes in the source task the better the Q-function that is learned. Note that for 800 episodes of the 3 vs. 2 task, the keepers are able to hold the ball for an average of 8.5 seconds, while for 1600 episodes their performance increases to 12.2 seconds. As the number of the training episodes in the source task increase the time that is required to reach the threshold decreases, showing that our method successfully improves performance in the target task.

The total time of the proposed approach in the target task is also less than the time-to-threshold without transfer learning in many cases. The best performance is 8.21

[5] We use 32 tilings for each variable. The width of each tile is set to 3 meters for the distance variables and 10 degrees for the angle variables. We set the learning rate, α, to 0.125, ϵ to 0.01 and λ to 0. These values remain fixed for all experiments.

3 vs. 2			4 vs. 3		
#episodes	train time	performance	time-to-thr.	total time	st. dev.
0	0	-	13.38	13.38	2.02
100	0.11	4.38	13.19	13.30	1.77
200	0.23	4.67	12.59	12.82	2.10
400	0.72	6.71	12.08	12.80	1.70
800	1.73	8.52	10.28	12.01	0.97
1600	4.73	12.20	3.48	**8.21**	1.16
2500	8.42	16.02	4.16	12.44	0.60
3200	12.17	16.84	**2.76**	14.95	0.28

Table 3. The table shows the training time and average performance in the source task, as well as time-to-threshold and total time in the target task. The best time-to-threshold and total time are in **bold**.

	source		5 vs. 4		
task	#episodes	tr. time	time-to-thr.	total time	st. dev.
-	0	0	26.30	26.30	2.85
3 vs. 2	1600	4.73	15.54	20.27	3.24
3 vs. 2	2500	8.42	8.88	17.31	3.23
3 vs. 2	3200	12.17	**3.43**	**15.60**	1.24
4 vs. 3	4000	7.52	10.07	17.59	1.49
4 vs. 3	6000	12.32	9.26	21.58	1.92

Table 4. Average training times (in hours) for 5 vs. 4. The results are averaged over 10 independent trials. The best time-to-threshold and total time are in **bold**.

hours, which corresponds to a reduction of 38.6% of the time-to-threshold without transfer learning. This performance is achieved when training the agents for 1600 training episodes in the source task. This result shows that rather than directly learning on the target task, it is actually faster to learn on the source task, use our transfer method, and only then learn on the target task.

In order to detect significant difference among the performances of the algorithms we use paired t-tests with 95% confidence. We perform seven paired t-tests, one for each pair of the algorithm without transfer learning and the cases with transfer learning. The test shows that the proposed approach is significantly better when it is trained with 800 and 1600 episodes.

Scaling up to 5 vs. 4 and 6 vs. 5 We also test the performance of the proposed approach in the 5 vs. 4 target task. The 5 vs. 4 threshold is set to 8.5 seconds. The 3 vs. 2 task with 1,600, 2,500 and 3,200 training episodes and the 4 vs. 3 task with 4,000 and 6,000 training episodes are used as source tasks. Table 4 shows the training times, time-to-threshold and their sum for the different source tasks and number of episodes averaged over 10 independent runs along with the standard deviation.

In all cases the proposed approach outperforms the no-transfer case. It is interesting to note that the best time-to-threshold is achieved, when using 3 vs. 2 as a source task, with fewer episodes than when using 4 vs. 3 as a source task. This means that a relatively simple source task may provide *more* benefit than a more complex source task. In addition, the 3 vs. 2 task requires less training time, as it is easier than the 4 vs. 3 task. We perform 5 paired t-tests, one for each case of the proposed approach against learning without transfer. The proposed method achieves statistically significantly higher performance (at the 95% confidence level) in all cases.

source task	#episodes	tr. time	6 vs. 5	total	st. dev.
-	0	0	45.66	45.66	4.77
3 vs. 2	1600	4.73	31.77	36.50	3.87
3 vs. 2	3200	12.17	**2.50**	**14.67**	0.27
4 vs. 3	4000	7.52	33.34	40.86	3.78
5 vs. 4	3500	5.17	45.38	50.55	2.92
5 vs. 4	8000	16.08	28.43	44.51	2.39

Table 5. Average training times (in hours) for 6 vs. 5. The results are averaged over 10 independent trials. The best time-to-threshold and total time are in **bold**.

Additionally, we also considered the 6 vs. 5 task, where the threshold is set to 8 seconds. As source tasks, we use the 3 vs. 2 tasks with 1600 and 3200 training episodes, the 4 vs. 3 task with 4000 training episodes and the 5 vs. 4 task with 3500 and 8000 episodes. Table 5 shows the results in a similar fashion to the previous table.

The best total time is achieved when a 3 vs. 2 task (trained for 3200 episodes) is used as the source task, reducing the total time roughly 68%. As in the case of 5 vs. 4, we again notice that when a simpler source task is used, both the time-to-threshold and the total training time decrease. This is an indication of the role of the use of the multiple mapping functions. The 6 vs. 5 game is decomposed to 25 5 vs. 4 instances to 100 4 vs. 3 instances and to 100 3 vs. 2 instances. In the second case the algorithm searches among a larger number of source instances and it is more likely to get better $Q-$values as more situations are considered in the phase of the source values transfer. Note that in the case of 5 vs. 4 (3500 episodes) as the source task, there is no improvement. These results demonstrate that the proposed approach scales well to large problems, especially when a small task is used as the source.

Besides the benefit of using multiple mappings with Value Addition compared to a single predefined mapping which requires domain knowledge, we further investigate if there are also performance benefits of using multiple mappings with Value-Addition in Keepaway. We compare the proposed approach with a variation that uses only one mapping function. We use the mapping function that corresponds to the first K_s keepers and T_s takers as they are ordered increasingly to their distance from the ball. This way we select the agents that are nearest to the learning agent and we shall refer to it as *nearest agents* (NA) mapping. We must mention here that the NA mapping is similar to the one that is used in [1].

We compare the two methods in the 4vs3 target task using the 3vs2 task as source. Table 6 shows the training times spent in 4vs3 task for different amounts of training episodes in the source task along with the total time averaged over ten independent repetitions. The last column shows the corresponding results of the proposed approach Multiple Mappings (MM) for comparison purposes.

The first observation is that in all cases MM outperforms NA. An interesting outcome is that NA in all cases did not succeeded to reduce the total training time. Additionally, in the case of 800 episodes we have negative transfer. We perform three paired t-tests between NA and MM for the different cases of training episodes in the source task. At a confidence level of 95% the tests show that MM performs significantly better than NA. The results indicate that using only a single mapping function is not adequate to enhance the learning procedure in the target task.

3vs2		4vs3-NA	4vs3-MM
#episodes	tr. time	total time	total time
0	0	13.38	13.38
800	1.73	15.52	12.01
1600	4.73	16.92	8.21
2500	8.42	18.63	12.44

Table 6. Average training times for 4vs3 of the NA algorithm for different number of training episodes in the source task averaged over 10 repetitions.

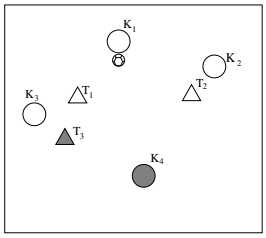

Fig. 3. Example of the NA mapping in the 4vs3 task.

An example that justifies this observation is given in Figure 3. The non-shaded agents are those considered by the NA mapping. In this case, K_1 can pass either to K_2 or K_3. Note that these keepers are blocked by takers T_1 and T_2 and ball interception is very likely. However K_4 is quite open, and a pass to K_4 would seem the appropriate action in this case. This shows that NA mapping is not always the best mapping in the source task and **that a single mapping can lead to inferior behaviour**.

6 Related Work

An approach that reuses past policies from solved source tasks to speed up learning in the target task is proposed in [9]. A restriction of this approach is that the tasks (source and target) must have the same action and state space. In contrast, our method allows the state and action spaces to be different between the target and the source task.

Advice based methods have been proposed in the past [10, 11]. [10] proposed a method that uses a list of learned rules from the source task as advices to the target task. The authors introduce three different utilization schemes of the advice. [11] export rules in first-order logic and translate them into advices.

A method presented in [1] and named *Transfer via inter-task mappings*, initializes the weights of the target task with the learned weights of the source task using mapping functions. The method depends strongly on the approximation function that is used in both tasks, as the function that transfers the weights is altered according to the approximation method. Additionally, a different approach is introduced in the same work which is named *Q-Value Reuse*. This method is similar to the way that we add the Q-values from the source task to the target task. The main difference is that in our method, we allow the use of multiple mappings and there is no need for a single mapping given beforehand by a domain expert.

An automatic method for constructing the mapping functions using exhaustive search on the set of all possible mappings was proposed in [7]. The main disadvantage of this method is that the computational complexity grows exponentially to the number of the state variables and actions.

In [12], each possible inter-state mapping is considered as an expert and with the use of a multi-armed bandit algorithm the agent decides which mapping to use. This method shows significant improvement but its performance is surpassed in the long run as it continues to explore always taking "advice" from low return experts.

Another promising method that has been proposed [13] is capable of learning a continuously parametrized function representing a stochastic mapping between the state spaces of two MDP's (soft homomorphism). However, this work doesn't handle action mappings and requires the explicit learning of a model in the source task and an on-line learning process of the mapping function, in the target task.

Finally there are other TL methods which are focused on *learning* a mapping and not on computationally discovering it like Value-Addition does. In an example of a work that aims to learn a mapping between arbitrary tasks [14], the authors use a data-driven method to discover a mapping between two tasks in a new dimensional space using sparse coding. However, the authors do not consider allowing multiple mappings and their method applies to model based RL agents as opposed to this work which applies to any value-based RL algorithm.

7 Conclusions

Concluding, this work presented a novel method for autonomous multiple-mappings selection in TL. Results in Mountain Car 3D and on several instances of Keepaway have shown that Value-Addition can successfully select and use multiple mappings and improve learning via transfer. Using different mappings in different state space regions was significantly beneficial in Keepaway. This result further indicates that in some domains the use of multiple mapping is not only a step towards a more autonomous transfer of knowledge, but that it can also help to achieve better performance compared to a single mapping, even if that mapping is considered an intuitively correct one. Furthermore the results presented in this work provide similar conclusions, regarding the efficacy of value function transfer, to those presented by Taylor et al. [1]. However, we emphasize that the proposed method is able to succeed without requiring that a human provides a single inter-task mapping, but instead may autonomously select multiple mappings from a large set, successfully improving the autonomy of TD transfer learning.

Two limitations of the proposed method are that: 1) it requires a set of mappings to be given beforehand, then it is able to select the best of them. However, this set can theoretically be exhaustive and it can be generated in an automatic way and 2) since Value-Addition belongs to a family of value-based transfer methods it assumes the similarity of two state-action pairs if both of these have high values in their parent tasks. However, as discussed in Section 3, this assumption is partially relaxed through the use of the mean-value transfer criterion.

The above let us conclude with some ideas for future work. First, new methods to autonomously construct or learn the inter-task mappings should be explored, second,

for less related transfer domains, TL algorithms should exploit other MDP similarity features to identify the correct mappings such as the task's dynamics etc. Finally, more work is needed to further investigate as to where and how multiple mappings transfer should be chosen compared to single-mapping methodologies, even if a correct mapping is known beforehand.

8 Acknowledgments

The authors thank the reviewers for their comments and insights. This work was supported in part by NSF IIS-1149917.

References

1. Taylor, M.E., Stone, P., Liu, Y.: Transfer learning via inter-task mappings for temporal difference learning. Journal of Machine Learning Research **8** (2007) 2125–2167
2. Fachantidis, A., Partalas, I., Taylor, M., Vlahavas, I.: Transfer learning via multiple inter-task mappings. In: Recent Advances in Reinforcement Learning. Volume 7188. Springer (2012) 225–236
3. Sutton, R.S., Barto, A.G.: Reinforcement Learning, An Introduction. MIT Press (1998)
4. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup-soccer keepaway. Adaptive Behavior **13**(3) (2005) 165–188
5. Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Machine Learning **22**(1-3) (1996) 123–158
6. Taylor, M.E., Jong, N.K., Stone, P.: Transferring instances for model-based reinforcement learning. In: Proc. of ECML. (2008) 488–505
7. Taylor, M.E., Kuhlmann, G., Stone, P.: Autonomous transfer for reinforcement learning. In: 7th AAMAS. (2008) 283–290
8. Stone, P., Kuhlmann, G., Taylor, M.E., Liu, Y.: Keepaway soccer: From machine learning testbed to benchmark. In: RoboCup-2005: Robot Soccer World Cup IX. (2006) 93–105
9. Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. In: 5th international joint conference on Autonomous agents and multiagent systems. (2006) 720–727
10. Taylor, M.E., Stone, P.: Cross-domain transfer for reinforcement learning. In: 24th international conference on Machine learning. (2007) 879–886
11. Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill acquisition via transfer learning and advice taking. In: 17 th European Conference on Machine Learning. (2005) 425–436
12. Talvitie, E., Singh, S.: An experts algorithm for transfer learning. In: 20th IJCAI. (2007) 1065–1070
13. Sorg, J., Singh, S.: Transfer via soft homomorphisms. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. AAMAS '09 (2009) 741–748
14. Ammar, H.B., Tuyls, K., Taylor, M.E., Driessen, K., Weiss, G.: Reinforcement learning transfer via sparse coding. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (June 2012)