An Introduction to Inter-task Transfer for Reinforcement Learning

Matthew E. Taylor
Department of Computer Science
Lafayette College
Easton, PA 18042
mtaylor@cs.lafayette.edu

Peter Stone
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712-1188
pstone@cs.utexas.edu

Abstract

Transfer learning has recently gained popularity due to the development of algorithms that can successfully generalize information across multiple tasks. This article focuses on transfer in the context of reinforcement learning domains, a general learning framework where an agent acts in an environment to maximize a reward signal. The goals of this article are to 1) familiarize readers with the transfer learning problem in reinforcement learning domains, 2) explain why the problem is both interesting and difficult, 3) present a selection of existing techniques that demonstrate different solutions, and 4) provide representative open problems in the hope of encouraging additional research in this exciting area.

Introduction

Agents, physical and virtual entities that interact with their environment, are becoming increasingly prevalent. However, if agents are to behave intelligently in complex, dynamic, and noisy environments, we believe that they must be able to learn and adapt. The *reinforcement learning* (RL) paradigm is a popular way for such agents to learn from experience with minimal feedback. One of the central questions in RL is how best to generalize knowledge to achieve the two goals listed above.

In reinforcement learning problems, agents sequentially observe their state and execute actions. The goal is to maximize a real-valued reward signal, which may be time-delayed. For example, an agent could learn to play a game by being told what the state of the board is, what the legal actions are, and then whether it wins or loses at the end of the game. However, unlike in supervised learning scenarios, the agent is never provided the "correct" action. Instead, the agent can only gather data by interacting with an environment, receiving information about the results its actions and the reward signal. RL is often used because of the framework's flexibility and due to the development of increasingly data-efficient algorithms.

RL agents learn by interacting with the environment, gathering data. If the agent is virtual and acts in a simulated environment, training data can be collected at the expense of computer time. However, if the agent is physical, or the agent must act on a "real-world" problem where the on-line reward is critical, such data can be expensive. For instance, a physical robot will degrade over time and must be replaced and an agent learning to automate a company's operations may lose money while training. When RL agents begin learning *tabula rasa*, mastering difficult tasks may be infeasible as they require significant amounts of data, even when using state-of-the-art RL approaches. There are many contemporary approaches to speed up "vanilla" RL methods. *Transfer learning* (TL) is one such technique.

Transfer learning is an umbrella term used when knowledge is generalized not just across a single distribution of input data, but when knowledge can be generalized across data drawn from multiple distributions or even different domains, potentially reducing the amount of data required to learn a new task. For instance, one may train a classifier on one set of newsgroup articles (e.g., rec.*) and then want the classifier to generalize to newsgroups with different subject material (e.g., transfer to sci.*) (Dai *et al.* 2007). An example of human-level transfer was discussed in a recent issue of AI Magazine (Bushinsky 2009):

> World [chess] champion [Mikhail] Tal once told how his chess game was inspired from watching ice hockey! He noticed that many of the hockey players had the habit of hanging around the goal even when the puck was far away from it .... He decided to try transferring the idea to his chess game by positioning his pieces around the enemy king, still posing no concrete threats.

The insight that knowledge may be generalized across different tasks has been studied for years in humans (c.f., Skinner (1953)) and in classification settings (c.f., Thrun (1996)), but has only recently gained popularity in the context of reinforcement learning.

As discussed later in this article, there are many possible goals for transfer learning. In the most common scenario, an agent learns in a *source task* and then this knowledge is used to bias learning in a different *target task*. The agent may have different knowledge representations and learning algorithms in the two tasks, and transfer acts as a conduit for transforming source task knowledge so that it is useful in the target task. It is also reasonable to frame TL as one agent learning in a source task and a different agent learning in the target task, where transfer is used to share knowledge between the two agents. In most cases the two are algorithmically equivalent, as we assume we are able to directly access the source task agent's knowledge so that direct copying into a "new" agent is equivalent to having the "old" agent switch tasks.

This article provides an introduction to the problem of reusing agents' knowledge so that it can be used to solve a different task, or (equivalently) to transfer knowledge between agents solving different tasks. This article aims to 1) introduce the transfer problem in RL domains, 2) provide a sense of what makes the problem difficult, 3) present a sample of existing transfer techniques, and 4) discuss some of the shortcomings in current work to help encourage additional research in this exciting area. Readers interested in a significantly more detailed treatment of existing works are directed to our survey article (Taylor & Stone 2009).

In the following section we provide background on the reinforcement learning problem that should be sufficient to understand the remainder of the article. Readers interested in a more in-depth treatment of RL are referred elsewhere (Kaelbling, Littman, & Moore 1996; Sutton & Barto 1998; Szepesvári 2009). The next section introduces a pair of motivating domains to be used throughout the article. Following that, the Dimensions of Transfer section discusses the many ways in which transfer algorithms can differ in terms of their goals and abilities. A Selection of Transfer Learning Algorithms presents a selection of transfer algorithms in the context of the example domains, providing the reader with a brief introduction to existing solution techniques. Related Paradigms contrasts other popular generalization and speed-up learning approaches in RL with transfer learning. Lastly, we conclude with a discussion of open questions.

Reinforcement Learning Background and Notation

Reinforcement learning problems are typically framed as *Markov decision processes* (MDPs) defined by the 4-tuple $\{S, A, T, R\}$, where the goal of the agent is to maximize a single real-valued reward signal. In this article, we define a *domain* to be a setting for multiple related *tasks*, where MDP and task are used interchangeably. An agent perceives the current *state* of the world $s \in S$ (possibly with noise). The agent begins in a start state (drawn from a subset of states, S_0). If the task is *episodic*, the agent executes actions in the environment until it reaches a terminal state (a state in the set S_{final}, which may be referred to as a *goal state* if the reward there is relatively high), at which point the agent is returned to a start state.

The set A describes the *actions* available to the agent, although not every action may be possible in every state. The *transition function*, $T : S \times A \mapsto S$, takes a state and an action and returns the state of the environment after the action is performed. Transitions may be non-deterministic, making the transition function a probability distribution function. The *reward function*, $R : S \mapsto \mathbb{R}$, maps every state to a real valued number, representing the reward achieved for reaching the state. An agent senses the current state, s, and typically knows A and what state variables comprise S; however, it is generally not given R or T.

A *policy*, $\pi : S \mapsto A$, fully defines how an agent interacts with the environment by mapping perceived environmental states to actions. The success of an agent is determined by how well it maximizes the total reward it receives in the long run while acting under some policy π. There are many possible approaches to learning such a policy, but this article will focus on *temporal difference* (TD) methods, such as *Q-learning* (Sutton 1988; Watkins 1989) and *Sarsa* (Rummery & Niranjan 1994; Singh & Sutton 1996), which learn to approximate $Q : S \times A \mapsto \mathbb{R}$. At any time during learning, an agent can choose to *exploit*, by selecting the action that maximizes $Q(s, a)$ for the current state, or to *explore*, by taking a random action. TD methods for RL are among the most simple and well understood, causing them to be used in many different transfer learning works; some transfer learning methods can be explained using TD algorithms but then combined with more complex RL algorithms in practice.

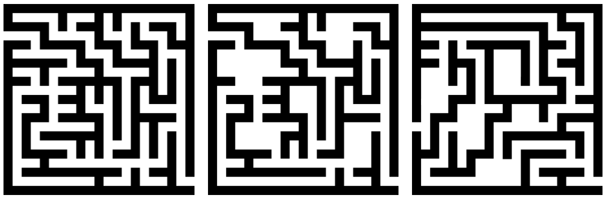
In tasks with small, discrete state spaces, Q and π can be fully represented in a table. As the state space grows, using a table becomes impractical, or impossible if the state space is continuous. TL methods are particularly relevant in such large MDPs, as these are precisely the problems that require generalization across states and may be particularly slow to learn. Agents in such tasks typically factor the state using *state variables* (or *features*), so that $s = \langle x_1, x_2, \ldots, x_n \rangle$. In such cases, RL methods use *function approximators*, such as artificial neural networks and tile coding, which rely on concise, parameterized functions and use supervised learning methods to set these parameters. Parameters and biases in the approximator define the state space abstraction, allowing observed data to influence a region of state-action values, rather than just a single state-action pair, and can thus substantially increase the speed of learning.

Two additional concepts used to improve the performance of RL algorithms will be referenced later in this article. The first, *options* (Sutton, Precup, & Singh 1999), are a type of temporal abstraction. Such macro-actions group a series of actions together and may be hand-specified or learned, allowing the agent to select actions that execute for multiple timesteps. The second, *reward shaping* (Mataric 1994), allows agents to use an artificial reward signal (R') rather than the MDP's true reward (R). For instance, a human could modify the reward of an agent in a maze so that it receives additional reward as it approached the goal state, potentially allowing the agent to learn faster.

Motivating Domains

This section provides two example RL domains where transfer has been successful, which will be used to discuss particular transfer methods later in this article. The first, maze navigation, contains many possible tasks, such as having different wall locations or goal states. In the second, robot soccer Keepaway, tasks typically differ in the number of players on each team, which results in different state representations and actions.

These two domains are selected because they have been used in multiple works on TL and because their properties compliment each other. Maze navigation tasks are typically single agent tasks situated in a fully observable, discrete state space. These relatively simple tasks allow for easy



(a) Maze 1 (b) Maze 2 (c) Maze 3

Figure 1: Three example mazes to motivate transfer learning

analysis of TL methods and represent a simplification of a number of different types of agent navigation tasks. Keepaway, in contrast, is a multi-agent task which models realistic noise in sensors and actuators, requiring significantly more data (i.e., time) to learn successful policies. Additionally, the number of state variables and number of actions change between tasks (as described below), introducing a significant challenge not necessarily seen in maze-like domains.

Maze Navigation

Learning to navigate areas or mazes is a popular class of RL problems, in part because learned policies are easy to visualize and to understand. Although the formulation differs from paper to paper, in general the agent operates in a discrete state space and can select from four movement actions {`MoveNorth`, `MoveSouth`, `MoveEast`, `MoveWest`}. The agent begins in a start state, or set of start states, and receives a positive reward upon reaching a goal state based on how many actions it has executed. The transition function is initially unknown and is determined by the location of the maze's walls.

As discussed later, multiple researchers have experimented with speeding up learning with TL by focusing on the structure of navigation tasks. For instance, a policy learned on a source task maze could be a useful when learning a target task maze with slightly different walls (e.g., transfer from Figure 1(a) to Figure 1(b)). Alternatively, it is often the case that navigating within rooms is similar, in that the goal is to efficiently traverse a relatively empty area in order to reach a doorway or corridor (e.g., transfer from Figure 1(b) to Figure 1(c)).

As mentioned above, there are many ways in which source and target tasks can differ, which dictate what abilities a transfer algorithm would need in order to be applicable. In the maze domain, for instance, differences could include:

- T: wall placement could change or the actions could be made stochastic
- S: the size of the maze could change
- S_0: the agent could start in different locations
- S_{final}: the goal state could move
- State variables: the agent's state could be an index into a discrete state space or could be composed of x, y coordinates
- R: the agent could receive a reward of -1 on every step or the reward could be a function of the distance from the goal state
- A: additional actions, such as higher-level macro actions could be enabled or disabled

Keepaway

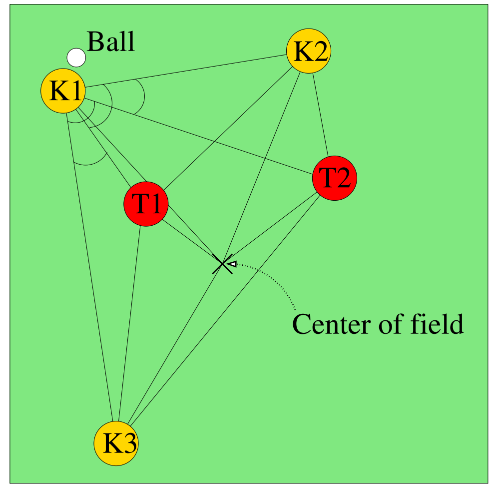


Figure 2: This diagram depicts the distances and angles used to construct the 13 state variables used in 3 vs. 2 Keepaway. Relevant objects are the 5 players, ordered by distance from the ball, and the center of the field.

The game of *Keepaway* is a subproblem in the RoboCup Soccer Server (Noda *et al.* 1998). A team of independently controlled *keepers* attempt to maintain possession of a ball in a square area while a team of *takers* try to gain possession of the ball or kick it out of bounds. The keeper with the ball chooses between macro actions, which depend on the number of keepers. The keepers receive a reward of +1 for every time step that the ball remains in play. The episode finishes when a taker gains control of the ball or the ball is kicked out of bounds. In the standard version of Keepaway, keepers attempt to learn to possess the ball as long as possible, increasing the average episode length, while the takers follow a fixed policy.

In 3 vs. 2 Keepaway (see Figure 2), there are three keepers and two takers. Rather than learning to execute low-level actuator movements, the keepers instead select from three hand-designed macro-actions: {`Hold`, $\texttt{Pass}_1$, $\texttt{Pass}_2$}, corresponding to "maintain possession," "pass to closest teammate," and "pass to second closest teammate." Keepers which do not possess the ball follow a hand-coded policy which attempts to capture an open ball or moves to get open for a pass. Learning which of the three actions to select on a given timestep is challenging because the actions are stochastic, the observations of the agents' state are noisy, and the 13-dimensional state (described in Table 1) is continuous (necessitating function approximation). This task has been used by multiple researchers, in part due to freely available players and benchmark performances distributed by UT-Austin (Stone *et al.* 2006).[1]

[1]For details and videos of play, see `http://cs.utexas.edu/~AustinVilla/sim/keepaway`

The 13 State Variables in 3 vs. 2 Keepaway

State Variable	Description
$dist(K_1, C)$	Distance from keeper with ball to center of field
$dist(K_1, K_2)$	Distance from keeper with ball to closest teammate
$dist(K_1, K_3)$	Distance from keeper with ball to second closest teammate
$dist(K_1, T_1)$	Distance from keeper with ball to closest taker
$dist(K_1, T_2)$	Distance from keeper with ball to second closest taker
$dist(K_2, C)$	Distance from closest teammate to center of field
$dist(K_3, C)$	Distance from second closest teammate to center of field
$dist(T_1, C)$	Distance from closest taker to center of field
$dist(T_2, C)$	Distance from second closest taker to center of field
Min($dist(K_2, T_1)$, $dist(K_2, T_2)$	Distance from nearest teammate to its nearest taker
Min($dist(K_3, T_1)$, $dist(K_3, T_2)$	Distance from second nearest teammate to its nearest taker
Min($ang(K_2, K_1, T_1)$, $ang(K_2, K_1, T_2)$	Angle of passing lane from keeper with ball to closest teammate
Min($ang(K_3, K_1, T_1)$, $ang(K_3, K_1, T_2)$	Angle of passing lane from keeper with ball to second closest teammate

Table 1: This table lists all state variables used for representing the state of 3 vs. 2 Keepaway. C is the center of the field, $dist(X, Y)$ is the distance between X and Y, and $ang(X, Y, Z)$ is the angle formed by X, Y, Z where X is the vertex. The state is ego-centric for the keeper with the ball and rotationally invariant.

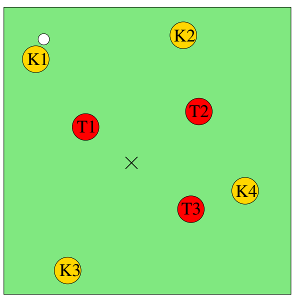


Figure 3: 4 vs. 3 Keepaway has 7 players in total and the keepers' state is composed of 19 state variables, analogous to 3 vs. 2 Keepaway.

If more players are added to the task, Keepaway becomes harder for the keepers because the field becomes more crowded. As more takers are added, there are more opponents to block passing lanes and chase down errant passes. As more keepers are added, the keeper with the ball has more passing options, but the average pass distance is shorter. This reduced distance forces more passes and often leads to more missed passes because of the noisy actuators and sensors. Additionally, because each keeper learns independently and can only select actions when it controls the ball, adding more keepers to the task means that keepers receive less experience per timestep. For these reasons, keepers in *4 vs. 3 Keepaway* (i.e., 4 keepers vs. 3 takers) require more simulator time to learn a good control policy, compared to 3 vs. 2 Keepaway.

In 4 vs. 3 Keepaway (see Figure 3), the reward function is very similar: there is a +1 to the keepers on every timestep the ball remains in play. Likewise, the transition function is similar because the simulator is unchanged. Now $A = \{\texttt{Hold}, \texttt{Pass}_1, \texttt{Pass}_2, \texttt{Pass}_3\}$, and each state is composed of 19 state variables due to the added players. It is also important to point out that the addition of an extra taker and keeper in 4 vs. 3 results in a qualitative change to the keepers' task. In 3 vs. 2, both takers must go towards the ball because two takers are needed to capture the ball from the keeper. In 4 vs. 3, the third taker is now free to roam the field and attempt to intercept passes. This changes the optimal keeper behavior, as one teammate is often blocked from receiving a pass by a taker.

TL results in the Keepaway domain often focus on transferring between different instances of the simulated robot soccer domain with different numbers of players. Many authors have studied transfer in this domain because 1) it was one of the first RL domains to show successful transfer learning results, 2) learning is non-trivial, allowing transfer to significantly improve learning speeds, and 3) different Keepaway tasks have well-defined differences in terms of both state variable and action sets. When transferring from 3 vs. 2 to 4 vs. 3, there are a number of salient changes that any TL algorithm must account for, such as:

- State variables: the agent's state variables must change to account for the new players
- A: there are additional actions when the number of possible pass receivers increases
- S_0: adding agents to the field causes them to start in different positions on the field

Dimensions of Transfer

We now turn our attention to ways in which transfer learning methods can differ, providing examples in the maze and Keepaway domains. The following sections will enumerate and discuss the different dimensions, both providing an overview of what is possible with transfer in RL and motivating how TL algorithms may differ. First, there are a number of different possible goals for transfer in RL domains. Second, the similarity of source and target tasks plays an important role in determining what capabilities a TL algorithm must have in order to be applied effectively (i.e., how the source task(s) is mapped to the target task in Figure 4(a)). Third, transfer methods can differ in terms of what RL methods they are compatible with (i.e., how the source task and target task agents learn their policies). Fourth, different TL

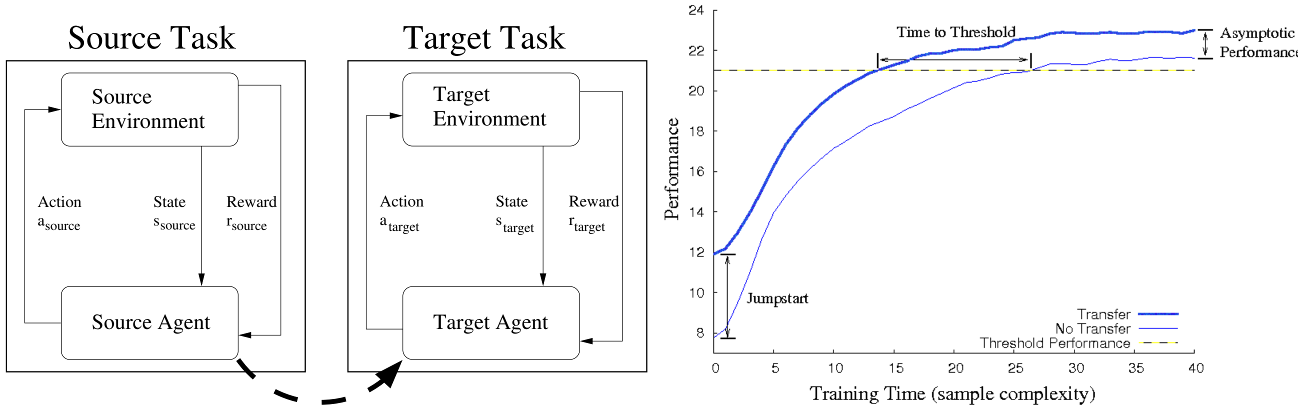


Figure 4: In general, transfer in RL can be described as transferring information between a source task agent to a target task agent (left). Many different metrics to evaluate TL algorithms are possible. This graph (right) shows benefits to the jumpstart, the asymptotic performance, the total reward (the area under a learning curve), the reward area ratio (the ratio of the area under the transfer to the area under the non-transfer learning curve), and the time to threshold.

methods transfer different types of knowledge (i.e, what is transferred between the source task and target task agents).

Possible Transfer Goals

Current transfer learning research in RL does not have a standard set of evaluation metrics, in part because there are many possible options. For instance, it is not always clear how to treat learning in the source task: whether to charge it to the TL algorithm or to consider it a “sunk cost.” One possible goal of transfer is to reduce the overall time required to learn a complex task. In this scenario, a *total time scenario*, which explicitly includes the time needed to learn the source task, would be most appropriate. Another reasonable goal of transfer is to effectively reuse past knowledge in a novel task. In such a *target task time scenario*, it is reasonable to only account for the time used to learn the target task. The majority of current research focuses on the second scenario, ignoring time spent learning source task(s) by assuming that the source task(s) have already been learned — the existing knowledge can either be used to learn a new task or simply ignored.

The degree of similarity between the source task and the target task will have markedly different effects on TL metrics depending on whether the a total time scenario or target task time scenario applies. For instance, suppose that the source task and target task were *identical* (thus technically not an instance of TL). In the target task time scenario, learning the source task well would earn the system high marks. However, in the total time scenario, there would be little performance change from learning on a single task. On the other hand, if the source and target task were unrelated, transfer would likely provide little benefit, or possibly even hurt the learner (i.e., cause *negative transfer*, as discussed later as a current open question).

Many metrics to measure the benefits of transfer in the target task are possible, such as those shown in Figure 4(b) (replicated from our past transfer learning work (Taylor & Stone 2007a)):

1. The **jumpstart** is the difference in initial performance of an agent using transfer, relative to learning without transfer.
2. The **asymptotic performance** (i.e., the agent’s final learned performance) may be changed via transfer.
3. The **total reward** accumulated by an agent (i.e., the area under the learning curve) may differ if it uses transfer, compared to learning without transfer.
4. The **transfer ratio** is defined as the total reward accumulated by the transfer learner divided by the total reward accumulated by the non-transfer learner.
5. The **time to threshold** measures the learning time (or samples) needed by the agent to achieve a pre-specified performance level.

Transfer may be considered a success if metric 1 is greater than zero, metrics 2–3 are increased with transfer, metric 4 is greater than one, or if metric 5 is reduced via transfer. While these metrics are domain independent, none are useful to directly evaluate the relative performance of transfer learning algorithms unless the tasks, learning methods, and representations are constant (e.g., it is generally not useful to directly compare the jumpstart of two different transfer learning algorithms if the algorithms were tested on different pairs of source and target tasks).

Improvement due to transfer from 3 vs. 2 Keepaway into 4 vs. 3 Keepaway could be measured, for example, by 1) improving the average episode time in 4 vs. 3 at the very beginning of learning in the target task, 2) learning to maintain possession of the ball for longer episodes in 4 vs. 3, 3) increasing the area under the 4 vs. 3 Keepaway learning curve when measuring the average episode vs. the simulator learning time, 4) increasing that same area under a 4 vs. 3 Keepaway learning curve relative to learning without transfer, and 5) reducing the time required by keepers to learn to reach an average episode length of 21.0 simulator seconds in 4 vs. 3 Keepaway.

Tasks Similarity

One way to categorize transfer methods is to consider how the source task and target tasks are related. For instance, are the two tasks similar enough for the learned source task knowledge to be directly used in the target task, or must the source task knowledge be modified before being used in the target task? Possible variants along this dimension are as follows.

- Knowledge is "copied" from a source task agent directly into a target task agent. Such methods are typically simple, but require the source and target tasks to be very similar. *Example*: a policy learned in a source maze is used directly in a target task maze.
- Knowledge from a source task agent is modified before being used by a target task agent. This modification leverages a hand-coded or learned relationship between the source and target task. Such methods enable the source and target tasks to differ significantly, but require knowledge about how the two tasks are similar. *Example*: a Q-value function is learned in 3 vs. 2 Keepaway, transformed to account for novel state variables and actions, and then used to initialize keepers in 4 vs. 3.
- The TL method *learns* how a pair of tasks are similar and then uses this similarity to modify source task agent knowledge for use by the target task agent. Such methods are much more flexible, but may be impractical in the general case. *Example*: rather than using human knowledge, an algorithm discovers how state variables and actions are similar in 3 vs. 2 and 4 vs. 3 Keepaway, and then transfers a state-action value function between the two tasks.

Relatedness of Agents

TL algorithms are often designed to be able to accommodate specific types of differences between the source task agents and target tasks agents, providing another dimension along which to classify TL algorithms. Do agents in the two tasks need to use the same learning method, the same class of learning methods, and/or the same knowledge representation? For example, a particular transfer algorithm may require one of the following relationships between source and target agents. Although each of the following options has been explored in the literature, the first is the most common.

- The source and target agents must use the same learning method. *Example*: keepers in 3 vs. 2 use Sarsa with tile coding and keepers in 4 vs. 3 use Sarsa with tile coding.
- The source and target agents may use different learning methods, but they must be similar. *Example*: keepers in 3 vs. 2 use Sarsa with tile coding and keepers in 4 vs. 3 use Q(λ) with tile coding. (Both Sarsa and Q-learning are temporal difference methods.)
- The source and target agents must use a similar learning method, and the representation can change. *Example*: an agent learns to navigate a source task maze using Sarsa with a Q-table (i.e., no function approximation) and then an agent learns to navigate a target task maze using Q(λ) with tile coding.
- The source and target agents may use different learning methods and representations. *Example*: keepers in 3 vs. 2 use Sarsa with tile coding and keepers in 4 vs. 3 use direct policy search with a neural network.

As with task similarity, TL methods which are more flexible are often more complex and/or require more data to be effective.

What Can be Transferred

Lastly, TL methods can be classified by the type of knowledge transferred and its specificity. Low-level knowledge, such as $\langle s, a, r, s' \rangle$ instances, an action-value function (Q), a policy (π), and a full model of the source task may all be transferred between tasks. Higher level knowledge, such as, partial policies (e.g., options), and shaping rewards (where a modified reward signal helps guide the agent) may not be directly used by the algorithm to fully define an initial policy in a target task, but such information may help guide the agent during learning.

All of these types of knowledge have been successfully transferred in maze domains and/or Keepaway domains. The benefits and tradeoffs of these different types of knowledge has not yet been examined, but we anticipate that different transfer situations will favor different types of transfer information.

A Selection of Transfer Learning Algorithms

One good way to gain a sense of the state-of-the-art is through examples of past successful results. This section provides a selection of such methods based on our two example domains. First, we investigate methods which act in maze or navigation domains. Second, we discuss methods that transfer between simulated robot soccer games. Third, we highlight two methods to learn relationships between tasks.

Table 2 gives a summary of how the transfer methods discussed in our two example domains differ. The first three methods copy knowledge directly between tasks without any modification. The second set of papers modify knowledge gained in a source task before using it in a target task. The third set learn how two tasks are related so that they may modify source task knowledge for use in a target task.

Salient differences between source and target tasks include having different start states, goal states, problem-spaces (described in the text), actions, reward functions, or state representations. When learning how tasks are similar, methods may rely on a qualitative description of the transition function, state variable groupings (groups), or experience gathered in the target task (exp), all of which are explained later in the text. Policies, reward functions, options, Q-value functions, rules, and $\langle s, a, r, s' \rangle$ instances may be transferred between the source task and target task. The metrics used are the total reward (tr), jumpstart (j), asymptotic performance (ap), and time to threshold (tt). All papers measure target task learning time (i.e., do not count time spent learning in the source task), and those marked with a † also measure the total learning time.

Paper	Task Differences	Transferred Knowledge	Knowledge to Learn Mapping	Metrics Used
Knowledge Copied: Maze Navigation Tasks				
(Fernandez & Veloso 2006)	start & goal states	π	N/A	tr
(Konidaris & Barto 2006)	problem-space	R	N/A	j, tr
Knowledge Modified: Simulated Robot Soccer				
(Taylor, Stone, & Liu 2007)	A, s-vars	Q	N/A	tt†
(Torrey *et al.* 2006)	A, R, s-vars	rules	N/A	j, tr
(Torrey *et al.* 2006)	A, R, s-vars	options	N/A	j, tr
(Taylor & Stone 2007a)	A, R, s-vars	rules	N/A	j, tr, tt†
Learning Transfer Mappings				
(Liu & Stone 2006)	A, s-vars	N/A	T (qualitative)	N/A
(Soni & Singh 2006)	A, s-vars	options	groups, exp	ap, j, tr
(Taylor, Jong, & Stone 2008)	A, s-vars	instances	exp	ap, tr

Table 2: This table provides a high-level overview of TL algorithms discussed in this section. Abbreviations are explained in the text.

Maze Navigation

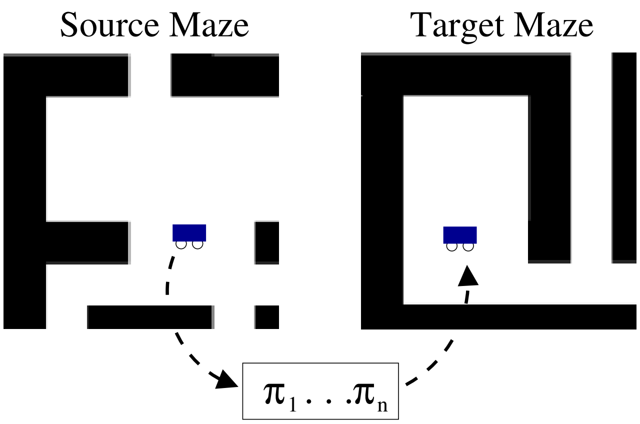


Figure 5: Many methods directly use a learned source task policy to initialize an agent's policy in the target task, or to bias learning in the target task.

To begin our discussion of current transfer learning methods, we examine a set of algorithms that tackle transfer in navigation tasks, e.g., by transferring a policy between tasks (see Figure 5). Both transfer methods discussed in this section assume that agents in the source and target task use the same TD learning algorithm and function approximators, and they only consider the target task time scenario.

Probabilistic Policy Reuse *Probabilistic policy reuse* (Fernandez & Veloso 2006) considers maze tasks in which only the goal state differs. In this work, the method allows a single goal state to differ between the tasks, but requires that S, A, and T remain constant. Action selection is performed by the *π-reuse Exploration Strategy*. When the agent is placed in a novel task, on every timestep, it can choose to: exploit a learned source task policy, randomly explore, or exploit the current learned policy for the target task. By setting these exploration parameters, and then decaying them over time, the agent balances exploiting past knowledge, improving upon the current task's policy by measuring its efficacy in the current task, or exploring in the hope of finding new actions that will further improve the current task's policy, eventually converging to a near-optimal policy. Results show that using the π-reuse Exploration to transfer from a single source task performs favorably to learning without transfer, although performance may be decreased when the source task and target task have goal states in very different locations. Performance is measured by the total reward accumulated in a target task.

In addition to transfer from a single source task, Fernández and Veloso's *PRQ-Learning* method is one of the few that can transfer from *multiple* source tasks (Fernandez & Veloso 2006). PRQ-Learning extends π-reuse exploration so that more than one past policy can be exploited. As past policies are exploited, their usefulness in the current task is estimated, and over time a softmax selection strategy (Sutton & Barto 1998) makes the most useful past policies most likely to be used. Lastly, the *Policy Library through Policy Reuse* algorithm allows learned policies to be selectively added to a policy library. Once a task is learned, the new policy can be added to the library if it is dissimilar from all policies existing in the library, according to a parameterized similarity metric. This metric can be tuned change the number of policies admitted to the library. Increasing the number of policies in the library increases the chance that a past policy can assist learning a new task, at the expense of increased policy evaluation costs in PRQ-Learning. This approach is similar in spirit to that of case-based reasoning, where the similarity or reuse function is defined by the parameterized similarity metric. In later work Fernández *et al.* (2010) extend the method so that it can be used in Keepaway as well.

Agent- and Problem-Space Rather than transfer policies from one task to another, Konidaris and Barto (2006) instead transfer a learned shaping function. A shaping function provides higher-level knowledge, such as "get rewarded for moving towards a particular beacon," which may transfer well when the source task and target task are so dissimilar that direct policy transfer would not be useful. In order to learn the shaping reward in the source task, the standard problem is separated into *agent-space* and *problem-*

space representations. The agent-space is determined by the agent's capabilities, which remain fixed (e.g., physical sensors and actuators), and the space may be non-Markovian[2]. For instance, the agent may have a sensor that is always able to determine the direction of a beacon, and this sensor may be used in any task the agent faces. The problem-space, on the other hand, may change between source and target problems and is assumed to be Markovian. The shaping reward is learned in agent-space, since this will be unchanged in the target task.

In order for this method to work, the authors acknowledge that the transfer must be between *reward-linked* tasks, where "the reward function in each environment consistently allocates rewards to the same types of interactions across environments." For instance, an agent may learn to approach a beacon in the source task and transfer this knowledge as a shaping reward, but if the target rewards the agent for moving *away* from a beacon, the transferred knowledge will be useless or possibly even harmful. Determining whether or not a sequence of tasks meet this criterion is currently unaddressed, but it may be quite difficult in practice. This problem of determining when transfer will and will not be effective is currently one of the more pressing open problems, which may not have a simple solution.

In our opinion, agent- and problem-space is an idea that should be further explored as they will likely yield additional benefits. For instance, discovering when tasks are "reward-linked" so that transfer will work well would make the methods more broadly applicable. Particularly in the case of physical agents, it is intuitive that agent sensors and actuators will be static, allowing information to be easily reused. Task-specific items, such as features and actions, may change, but should be faster to learn if the agent has already learned something about its unchanging agent-space.

Keepaway

In this section we discuss a selection of methods that have been designed to transfer knowledge between Keepaway tasks. As discussed earlier, this can be a particularly difficult challenge, as the state variables and actions differ between different versions of Keepaway. All methods in this section require that the target task is learned with a TD learner.

We will first introduce inter-task mappings, a construct to enable transfer between agents that act in MDPs with different state variables and actions. We then discuss a handful of methods that use inter-task mappings to transfer in the simulated robot soccer domain.

Inter-task Mapping In order to enable TL methods to transfer between tasks that do have such differences, the agent must know how the tasks are related. This section provides a brief overview of *inter-task mappings* (Taylor, Stone, & Liu 2007), one formulation of task mappings. Such mappings can be used by all methods discussed in this section and can be thought of as similar to how case-based reasoning decides to reuse past information.

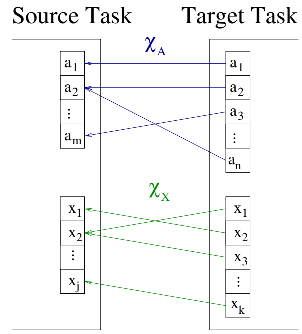


Figure 6: χ_A and χ_X are independent mappings that describe similarities between two MDPs. These mappings describe how actions in the target task are similar to actions in the source task and how state variables in the target task are similar to state variables in the source task, respectively.

To transfer effectively, when an agent is presented with a target task that has a set of actions, A', it must know how those actions are related to the action set in the source task, A. (For the sake of exposition we focus on actions, but an analogous argument holds for state variables.) If the TL method knows that the two action sets are identical, no action mapping is necessary. However, if this is not the case, the agent needs to be told, or learn, how the two tasks are related. One solution is to define an *action mapping*, χ_A, such that actions in the two tasks are mapped so that their effects are "similar," where similarity is loosely defined so that it depends on how the action affects the agent's state and what reward is received.[3] Figure 6 depicts an action mapping as well as a *state-variable mapping* (χ_X) between two tasks. A second option is to define a *partial mapping* (Taylor, Whiteson, & Stone 2007), such that any novel actions or novel state variables in the target task are not mapped to anything in the source task (and are thus ignored). Because inter-task mappings are not functions, they are typically assumed to be easily invertible (i.e., mapping source task actions into target task actions, rather than target task actions to source task actions).

For a given pair of tasks, there could be many ways to formulate inter-task mappings. Much of the current TL work assumes that a human has provided a (correct) mapping to the learner. In the Keepaway domain, we are able to construct inter-task mappings between states and actions in the two tasks based on our domain knowledge. The choice for the mappings is supported by empirical evidence showing that using these mappings do allow us to construct transfer functions that successfully reduce training time.

We define χ_A, the inter-task action mapping for the two tasks, by identifying actions that have similar effects on the

[2] A standard assumption is that a task is Markovian, meaning that the probability distribution over next states is independent of the agent's state and action history. Thus, saving a history would not assist the agent when selecting actions, and it can consider each state in isolation.

[3] An inter-task mapping often maps multiple entities in the target task to single entities in the source task because the target task is more complex than the source, but in general the mappings may be one-to-many, one-to-one, or many-to-many.

χ_A: 4 vs. 3 Keepaway to 3 vs. 2 Keepaway

4 vs. 3 Action	3 vs. 2 Action
`Hold Ball`	`Hold Ball`
$\texttt{Pass}_1$	$\texttt{Pass}_1$
$\texttt{Pass}_2$	$\texttt{Pass}_2$
$\texttt{Pass}_3$	$\texttt{Pass}_2$

Table 3: This table describes the (full) action mapping from 4 vs. 3 to 3 vs. 2.

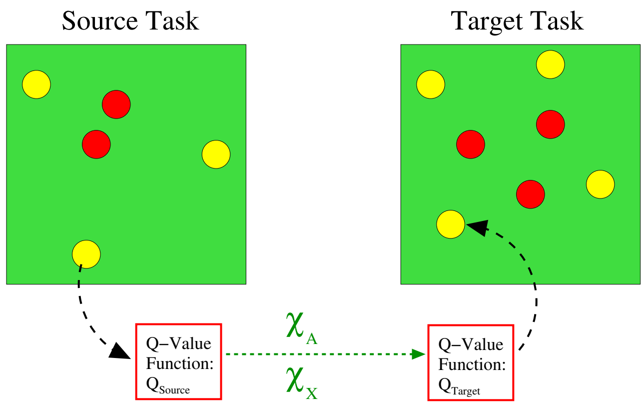


Figure 7: In Value Function Transfer, learned Q-values from the source task are transformed via inter-task mappings. These transformed Q-values are then used to initialize agents in the target task, biasing their learning with low-level knowledge from the source task.

world state in both tasks (see Table 3). For the 4 vs. 3 and 3 vs. 2 Keepaway tasks, the action "Hold ball" is equivalent because this action has a similar effect on the world in both tasks. Likewise, the first two pass actions are analogous in both tasks. We map the novel target action "Pass to third closest keeper" to "Pass to second closest keeper" in the source task. To define a partial action mapping between 4 vs. 3 and 3 vs. 2, the "Pass to third closest keeper" 4 vs. 3 action would be ignored, as there is no direct correspondence in 3 vs. 2. The state variable mapping for 4 vs. 3 to 3 vs. 2 can likewise be defined (Taylor, Stone, & Liu 2007).

Value Function Transfer Value Function Transfer (Taylor, Stone, & Liu 2007) differs from the methods previously discussed as it makes use of inter-task mappings, as well as transferring a different type of knowledge. Rather than transferring a policy or even a shaping reward, the authors extract Q-values learned in the source task and then use hand-coded inter-task mappings to transfer learned action-value functions from one Keepaway task to another (see Figure 7). It may seem counterintuitive that low-level action-value function information is able to speed up learning across different tasks — rather than using abstract domain knowledge, this method transfers very task-specific values. When this knowledge is used to initialize the action-value functions in a different keepaway task, the agents receive a small jumpstart, if any, because the Q-values are not "correct" for the new task. However, as experiments show, the transferred knowledge biases the

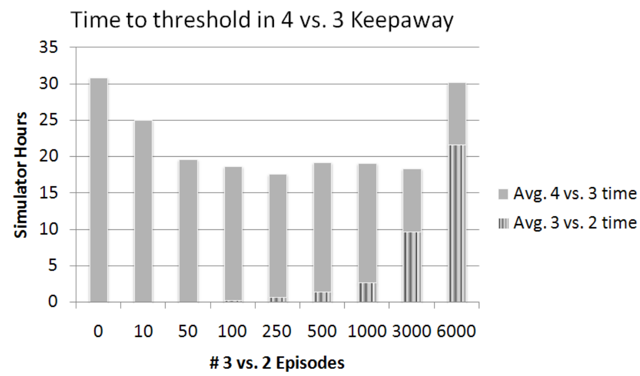


Figure 8: This bar chart highlights some results from Value Function Transfer work and shows the total number of simulator hours of training was needed to achieve a threshold performance (11.5 simulator seconds/episode). The x-axis shows the number of episodes spent training in 3 vs. 2 and the y-axis reports the total number of simulator hours spent training, separating out time spent in the 3 vs. 2 source task and the 4 vs. 3 target task.

learners so that they are able to learn faster.

Results from transfer from 3 vs. 2 Keepaway to 4 vs. 3 Keepaway are shown in Figure 8, which compares learning without transfer to eight different transfer learning scenarios. The x-axis denotes the number of episodes spent training in the 3 vs. 2 task, and the y-axis records the total simulator time needed to meet a specified performance threshold in the 4 vs. 3 task (an average of 11.5 seconds per episode). The bar on the far left shows that the average time needed to achieve the threshold performance using only target task learning is just over 30 simulator hours. On the far right, agents train in the source task for 6,000 episodes (taking just over 20 hours of simulator training time), but then are able to train in the target task for just under 9 simulator hours, significantly reducing the time total required to reach the target task threshold. In the Keepaway domain, spending increased amounts of time in the 3 vs. 2 source task generally decreases the amount of time needed to train the 4 vs. 3 target task. This result shows that if one's goal is to reduce target task training time, it can indeed make sense to use past knowledge, even if from a task with different state variables and actions.

Finally, consider when keepers train for 250 episodes in 3 vs. 2. In this scenario, the total training time is reduced relative to no transfer. This result shows that in some cases it may be beneficial to train on a simple task, transfer that knowledge, and then learn in a difficult task, rather than directly training on the difficult task. The authors suggest that in order to reduce the total training time, it may be advantageous to spend enough time to learn a rough value function, transfer into the target task, and then spend the majority of the training time refining the value function in the target task.

However, Value Function Transfer is not a panacea; the authors also admit that transfer may fail to improve learning. For instance, consider the game of *Giveaway*, where 3 agents must try to lose the ball as fast as possible. The authors show that transfer from Giveaway to Keepaway can be harmful, as might be expected, as the transferred action-value function is not only far from optimal, but it also provides an incorrect bias.

The authors provide no guarantees about their method's effectiveness but do hypothesize conditions under which their TL method will and will not perform well. Specifically, they suggest that their method will work when one or both of the following conditions are true:

1. The best learned actions in the source task, for a given state, are mapped to the best action in the target task via the inter-task mappings.
2. The average Q-values learned for states are of the correct magnitude in the trained target task's function approximator.

The first condition biases the learner to select the best actions more often in the target task, even if the Q-values are incorrect. For instance, `Hold` is often the best action to take in 3 vs. 2 Keepaway and 4 vs. 3 Keepaway if no taker is close to the keeper with the ball. The second improves learning by requiring fewer updates to reach accurate Q-values. The second condition is true when reward functions and learned performances are similar, as is true in different keepaway tasks.

Higher-Level Transfer Approaches We now turn our attention to methods that transfer higher-level advice than an action-value function. Our intuition is that such domain-level information should transfer better than low-level, task-specific information, but no empirical or theoretical studies have examined this particular question.

Torrey *et al.* (2006) introduce the idea of transferring *skills* between different tasks, which are similar to options. Inductive logic programming is used to identify sequences of actions that are useful most to agents in a source task. These skills are then mapped by human-supplied inter-task mappings into the target task, where they are used to bootstrap reinforcement learning. Torrey's subsequent work (Torrey *et al.* 2007) further generalizes the technique to transfer *relational macros*, or *strategies*, which may require composing several skills together. Inductive logic programming is again used to identify sequences of actions that result in high reward for the agents, but now the sequences of actions are more general (and thus, potentially, more useful for agents in a target task).

Once the strategies are re-mapped to the target task via a human-provided mapping, they are used to demonstrate a strategy to the target task learner. Rather than explore randomly, the target task learner always executes the transferred strategies for the first 100 episodes and thus learns to estimate the Q-values of the actions selected by the transferred strategies. After this demonstration phase, the strategies are discarded, having successfully seeded the agent's action-value function. For the remainder of the experiment, the agent selects from the MDP's base actions, successfully learning to improve on the performance of the transferred strategies.

One particularly interesting challenge the authors tackle in their papers is to consider source tasks and target tasks

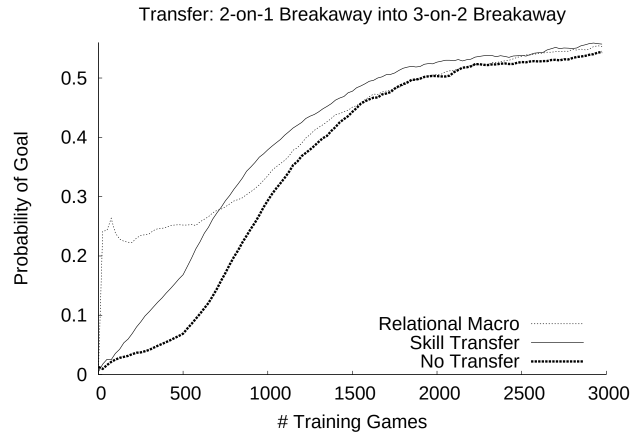


Figure 9: This graph shows one set of results in a simulated soccer goal scoring task comparing relational macro transfer, skill transfer, and learning without transfer.

that not only differ in terms of state variables and actions, but also in reward structure. Positive transfer is shown between 3 vs. 2 Keepaway, 3 vs. 2 MoveDownfield, and different versions of BreakAway. MoveDownfield is similar to Keepaway, except that the boundary area shifts over time (similar to how players may want to maintain possession of a ball while advancing towards an opponent's goal). BreakAway is less similar, as the learning agents now can pass to a teammate or shoot at the goal — rather than a reward based on how long possession is maintained, the team reward is binary, based on whether a goal is scored. Figure 9) reports one set of results, showing that using relational macros significantly improves the jumpstart and total reward in the target task when transferring between different BreakAway tasks. The simpler method, skill transfer, also improves learning performance, but to a lesser degree, as the information transferred is less complex (and, in this case, less useful for learning the target task).

****Add a sentence defining skill transfer algorithm in Fig 9****

Similar in spirit to relational macro transfer, our past work introduced *rule transfer* (Taylor & Stone 2007a) which uses a simple propositional rule learning algorithm to summarize a learned source task policy (see Figure 10). In particular, the learning algorithm examines (state, action) pairs from the source task agent and generates a decision list to approximate the agent's policy (i.e., the mapping from states to actions). In addition to again allowing high-level knowledge to be transferred between tasks, this work is novel because it looks at transferring knowledge between different domains, producing substantial speedups when the source task is orders of magnitude faster to learn than the target task.

The learned propositional rules are transformed via inter-task mappings so that they apply to a target task with different state variables and actions. The target task learner may then bootstrap learning by incorporating the rules as an extra action, essentially adding an ever-present option "take the action suggested by the source task policy," re-

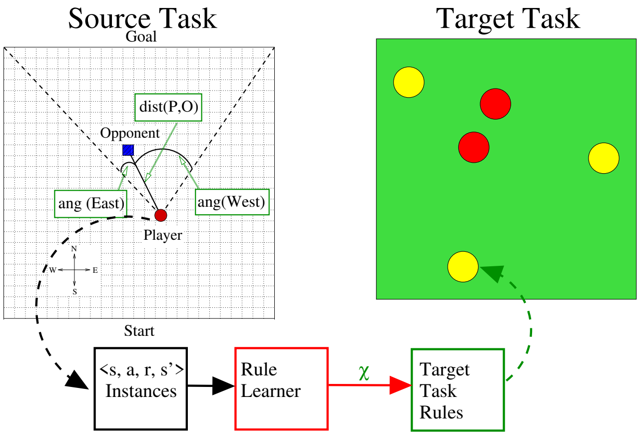


Figure 10: Inter-domain transfer can result in significant improvements to the total learning time required. For instance, the Knight's Task is a simple game that can be learned optimally in less than 30 seconds, which can be used to speed up learning in Keepaway, saving hours of training time.

sulting in an improved jumpstart, total reward, and time to threshold. By using rules as an intermediary between the two tasks, the source and target tasks can use different learning methods, effectively de-coupling the two learners. Two different source tasks are hand constructed: Ringworld and Knight Joust. Both tasks can be learned optimally in a matter of seconds, but the knowledge gained by agents in these simple tasks could still significantly improve learning rates in 3 vs. 2 Keepaway, reducing both the target task learning time and total learning time, relative to learning 3 vs. 2 Keepaway. For instance, high-level concepts like "perform a knight jump if the opponent gets close, otherwise move forward" can transfer well into Keepaway as "perform the pass action if a keeper gets close, otherwise hold the ball."

Learning Inter-task Mappings

The transfer algorithms considered thus far have assumed that a hand-coded mapping between tasks was provided, or that no mapping was needed. In this section we consider the less-well explored question of how an inter-task mapping can be learned, a critical capability if a human is unable, or unwilling, to provide a mapping. There are two high-level approaches to learning task similarity. The first relies on high-level structure of the tasks and the second uses low-level data gathered from agents in both tasks (see Figure 11 for an example data-driven approach, discussed later in this section).

The first category of techniques uses high-level information about tasks to reason about their similarity. If the full model of both MDPs were known (i.e., agents can plan a policy without exploring), a careful analysis of the two MDPs can reveal similarities that may be exploited. More relevant to this article is when some information about the source task and the target task is known, but a full model is not available. For instance, Liu and Stone (2006) assume that a type of qualitative model is available for learning a mapping, but such a model is insufficient for planning a policy. An algorithm based on the *Structure Mapping Engine* (Falkenhainer, Forbus, & Gentner 1989) is used to learn similarities between the tasks' qualitative models. The structure mapping engine is an algorithm that attempts to map knowledge from one domain into another using analogical matching — it is exactly these learned similarities that can be used as inter-task mappings to enable transfer.

The second category, data-driven learners, do not require a high-level model of tasks, allowing for additional flexibility and autonomy. Soni and Singh (2006) develop a TL method that requires much less background knowledge than the previous method, needing only information about how state variables are used to describe objects in a multi-player task.[4] First, an agent is supplied with a series of possible state transformations and an inter-task action mapping. Second, after learning the source task, the agent's goal is to learn the correct transformation: in each target task state s, the agent can randomly explore the target task actions, or it may choose to take the action recommend by the source task policy using a mapping, $\pi_{source}(X(s))$. This method has a similar motivation to that of Fernández and Veloso (2006), but here the authors learn to select between possible mappings rather than possible previous policies. Over time the agent uses Q-learning in Keepaway to select the best state variable mapping (χ_X), as well as learn the action-values for the target task (χ_A). The jumpstart, total reward, and asymptotic performance are all slightly improved when using this method, but its efficacy will be heavily dependant on the number of possible mappings between any source and target task. As the number of possibilities grows, so too will the amount of data needed to determine which mapping is most correct.

The MASTER algorithm (Taylor, Jong, & Stone 2008) further relaxes the knowledge requirements for learning an inter-task mapping: no knowledge about how state variables are related to objects in the target task is required. This added flexibility, however, comes at the expense of additional computational complexity. The core idea of MASTER is to 1) record experienced source task instances, 2) build an approximate transition model from a small set of experienced target task instances, and then 3) test possible mappings offline by measuring the prediction error of the target-task models on source task data (see Figure 11). This approach is sample efficient but has a high computational complexity, growing exponentially as the number of state variables and actions increase.

The main benefit of such a method is that this high computational cost occurs "between tasks," after the source task has been learned but before the target task agent begins learning, and thus does not require any extra interactions with the environment. The current implementation uses an exhaustive search to find the inter-task mappings that minimize the prediction error, but more sophisticated (e.g., heuristic) search methods could also be incorporated. It is

[4] For instance, an agent may know that a pair of state variables describe "distance to teammate" and "distance from teammate to marker," but the agent is not told *which* teammate the state variables describe.

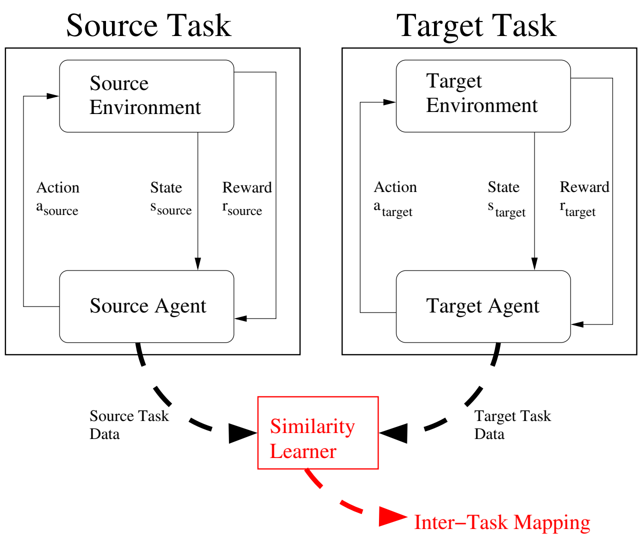


Figure 11: One class of methods to learn inter-task mappings rely on collecting data from (i.e., allowing an agent to interact with) source and target tasks. Similarities in the transitions and/or rewards in the two tasks are used to learn state variable and action mappings.

worth noting that MASTER is one of the few methods that transfers information when using a model-based learning method (rather than a model-free TD method). This method may also be used to assist with source task selection, but MASTER's primary contribution is to demonstrate that fully autonomous transfer is possible.

Open Questions

This section discusses a selection of current open questions in transfer for reinforcement learning domains. Having discussed a number of successful methods in TL, this section will help the reader better understand where TL may be further improved and we hope that it will inspire others to examine problems in this area. Many people have demonstrated empirical successes with transfer learning, but we discuss how 1) related paradigms may offer insight into improving transfer, 2) there are currently few theoretical results or guarantees for TL, 3) current TL methods are not powerful enough to completely remove a human from the loop, and 4) there are many ways the current TL methods could be more efficient.

Related Paradigms

Thus far, we have focused on algorithms that use one or more source tasks to better learn in a different, but related, target task. There are many other paradigms that may also be combined with, or provide ideas for, transfer learning. This section provides a brief discussion of these, as well as their relationship to TL.

Lifelong Learning Thrun (1996) suggested the notion of lifelong learning where an agent may experience a sequence of temporally or spatially separated tasks. Transfer would be a key component of any such system, but the lifelong learning framework is more demanding than that of transfer. For instance, transfer algorithms are typically "told" when a new task has begun, whereas in lifelong learning, agents may be reasonably expected to automatically identify new sub-tasks within the global environment (i.e., the real world). Transfer learning algorithms could, in theory, be adapted to automatically detect such task changes, similar to the idea of concept drift (Widmer & Kubat 1996).

Domain-Level Advice There is a growing body of work integrating advice into RL learners, which is often provided by humans (c.f., Maclin and Shavlik (1996) or Kuhlmann *et al.* (2004)). More relevant to this article are those methods which automatically extract domain-advice, effectively speeding up future tasks within a given domain (Torrey *et al.* 2006).

Reward Shaping *Reward shaping* in an RL context typically refers to allowing agent to train on an artificial reward signal (R') rather than the MDP's true reward (R). While there have been notable cases of humans providing shaping rewards, most important are methods which can learn domain-level shaping rewards (Konidaris & Barto 2006), representing a bias that may transfer well across multiple tasks. Another option that has not been explored in the literature, to the best of our knowledge, would be to shape the source task so that 1) the agent learned faster and reduced the total learning time, or 2) the source task became more similar to the target task, improving how useful the transferred information would be to the target task agent.

Representation Transfer Transfer learning problems are typically framed as leveraging knowledge learned on a source task to improve learning on a related, but different, target task. In other work, we (Taylor & Stone 2007b) examine the complimentary task of transferring knowledge between agents with different internal *representations* (i.e., the function approximator or learning algorithm) of the *same* task. Allowing for such shifts in representation gives additional flexibility to an agent designer as well as potentially allowing agents to achieve higher performance by changing representations partway through learning. Selecting a representation is often key for solving a problem (c.f., the *mutilated checkerboard problem* (McCarthy 1964) where humans' internal representations of a problem drastically changes the problem's solvability) and different representations may make transfer more or less difficult. Representation selection is a difficult problem in RL in general and we expect that the flexibility provided by representation transfer may enhance existing transfer learning algorithms.

Theoretical Advances

In addition to making progress in empirical enquires, attempting to better understand where and how TL will provide benefits, it is important to continue advancing theoretical understanding. For example, the majority of TL work in the literature has concentrated on showing that a particular transfer approach is plausible. None, to our knowledge, has a well-defined method for determining *when* an approach will succeed or fail according to one or more metrics. While we can say that it is possible to improve learning in a target task faster via transfer, we cannot currently decide if an arbitrary pair of tasks are appropriate for a given transfer method. Therefore, transfer may produce incorrect learning biases and result in negative transfer. While some methods can estimate task similarity (Taylor, Jong, & Stone 2008), these methods do not provide any theoretical guarantees about its effectiveness.

One potential method for avoiding negative transfer is to leverage the ideas of *bisimulation* (Milner 1982). Ferns *et al.* (2006) point out that:

> In the context of MDPs, bisimulation can roughly be described as the largest equivalence relation on the state space of an MDP that relates two states precisely when for every action, they achieve the same immediate reward and have the same probability of transitioning to classes of equivalent states. This means that bisimilar states lead to essentially the same long-term behavior.

However, bisimulation may be too strict because states are either equivalent or not, and may be slow to compute in practice. The work of Ferns *et al.* (2005; 2006) relaxes the idea of bisimulation to that of a (pseudo)metric that can be computed much faster, and gives a similarity measure, rather than a boolean. It is possible, although not yet shown, that bisimulation approximations can be used to discover regions of state space that can be transferred from one task to another, or to determine how similar two tasks are *in toto*. Such an approach may in fact be similar to that of analogical reasoning (e.g., the Structure Mapping Engine), but to our knowledge such a comparison has not been closely investigated.

Homomorphisms (Ravindran & Barto 2002) are a different abstraction that can define transformations between MDPs based on transition and reward dynamics, similar in spirit to inter-task mappings, and have been used successfully for transfer (Soni & Singh 2006). However, discovering homomorphisms is NP-hard (Ravindran & Barto 2003) and homomorphisms are generally supplied to a learner by an oracle. While these two theoretical frameworks may be able to help avoid negative transfer, or determine when two tasks are "transfer compatible," significant work needs to be done to determine if such approaches are feasible in practice, particularly if the agent is fully autonomous (i.e., is not provided domain knowledge by a human) and is not provided a full model of the MDP.

With a handful of exceptions (c.f., Bowling and Veloso (1999)), there has been relatively little work on the theoretical properties of transfer in RL. For example, there is considerable need for analysis that could potentially:

- Provide guarantees about whether a particular source task can improve learning in a target task (given a particular type of knowledge transfer).
- Correlate the amount of knowledge transferred (e.g., the number of samples) with the improvement in the source task.
- Define what an optimal inter-task mapping is, and demonstrates how transfer efficacy is impacted by the inter-task mapping used.

Autonomous Transfer

To be fully autonomous, an RL transfer agent would have to perform all of the following steps:

1. Given a target task, select an appropriate source task or set of tasks from which to transfer.
2. Learn how the source task(s) and target task are related.
3. Effectively transfer knowledge from the source task(s) to the target task.

While the mechanisms used for these steps will necessarily be interdependent, TL research has focused on each independently, and no TL methods are currently capable of robustly accomplishing all three goals. In particular, successfully performing all three steps in an arbitrary setting may prove to be "AI-Complete," but achieving the goal of autonomous transfer in at least some limited settings will improve the ability of agents to quickly learn in novel or unanticipated situations.

Optimizing Transfer

Recall that in Value Function Transfer (Figure 8), the amount of source task training was tuned empirically for the two competing goals of reducing the target task training time and the total training time. No work that we are aware of is able to determine the optimal amount of source task training to minimize the target task training time or total training time. It is likely that a calculation or heuristic for determining the optimal amount of source task training time will have to consider the structure of the two tasks, their relationship, and what transfer method is used. This optimization becomes even more difficult in the case of multi-step transfer, as there are two or more tasks that can be trained for different amounts of time.

Another question not (yet) addressed is how to best explore in a source task if the explicit purpose of the agent is to speed up learning in a target task. For instance, it may be better to explore more of the source task's state space than to learn an accurate action-value function for only part of the state space. While no current TL algorithms take such an approach, there has been some work on the question of learning a policy that is exploitable (without attempting to maximize the on-line reward accrued while learning) in non-transfer contexts (Şimşek & Barto 2006).

Finally, while there has been success creating source tasks by hand (Taylor & Stone 2007a), there are currently no methods for *automatically* constructing a source task given a target task. Furthermore, while multi-step transfer has proven successful, there are currently no guidelines on how to select/construct a single source task, let alone a series of source tasks. In cases where minimizing the total training time is critical, it may well be worth the effort to carefully design a *curriculum* (Taylor 2009) or *training regimen* (Zang *et al.* 2010) of tasks for the agent to sequentially learn, much as Skinner trained dogs to perform tricks through a series of tasks with increasing difficulty (Skinner 1953). Likewise, if a fully autonomous agent discovers a new task which is particularly hard, the agent may decide to first train on an artificial task, or series of tasks, before attempting to master the full task.

Conclusion

In the coming years, deployed agents will become increasingly common and gain more capabilities. Transfer learning, paired with reinforcement learning, is an appropriate paradigm if the agent must take sequential actions in the world and the designers do not know optimal solutions to the agent's tasks at design time (or even if they do not know what the agent's tasks will be). There are many possible settings and goals for transfer learning, but RL-related approaches generally focus on increasing the performance of learning and therefore potentially making RL more appropriate for solving complex, real-world systems.

Significant progress on transfer for RL domains has been made in the last few years, but there are also many open questions. We expect that many of the above questions will be addressed in the near future so that TL algorithms become more powerful and more broadly applicable. Additionally, we hope to see more physical and virtual agents utilizing transfer learning, further encouraging growth in an exciting and active subfield of the AI community.

Through this article, we have tried to provide an introduction to transfer learning in reinforcement learning domains. We hope that this article has served to better formulate the TL problem within the community, and to highlight some of the more prominent approaches. Additionally, given the number of open questions currently facing the field, we hope that others will be motivated to join in research of this exciting area.

Acknowledgements

We thank editor and reviewers for useful feedback and suggestions, as well as Lisa Torrey for sharing her data for Figure 9. This work has taken place in part at the Learning Agents Research Group (LARG) at the Artificial Intelligence Laboratory, The University of Texas at Austin. LARG research is supported in part by grants from the National Science Foundation (CNS-0615104 and IIS-0917122), ONR(N00014-09-1-0658), DARPA (FA8650-08-C-7812), and the Federal Highway Administration (DTFH61-07-H-00030).

References

[Argall *et al.* 2009] Argall, B.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469 – 483.

[Bowling & Veloso 1999] Bowling, M. H., and Veloso, M. M. 1999. Bounding the suboptimality of reusing subproblem. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1340–1347.

[Bushinsky 2009] Bushinsky, S. 2009. Deus ex machina — a higher creative species in the game of chess. *AI Magazine* 30(3):63–70.

[Caruana 1997] Caruana, R. 1997. Multitask learning. *Machine Learning* 28:41–75.

[Dai *et al.* 2007] Dai, W.; Yang, Q.; Xue, G. R.; and Yu, Y. 2007. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*.

[Falkenhainer, Forbus, & Gentner 1989] Falkenhainer, B.; Forbus, K. D.; and Gentner, D. 1989. The structure mapping engine: algorithm and examples. *Artificial Intelligence* 41:1–63.

[Fernandez & Veloso 2006] Fernandez, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems*.

[Fernández, García, & Veloso 2010] Fernández, F.; García, J.; and Veloso, M. 2010. Probabilistic policy reuse for inter-task transfer learning. *Journal of Robotics and Autonomous Systems*. to appear.

[Ferns *et al.* 2006] Ferns, N.; Castro, P.; Panangaden, P.; and Precup, D. 2006. Methods for computing state similarity in Markov decision processes. In *Proceedings of the 22nd Conference on Uncertainty in Artificial intelligence*, 174–181.

[Ferns, Panangaden, & Precup 2005] Ferns, N.; Panangaden, P.; and Precup, D. 2005. Metrics for Markov decision processes with infinite state spaces. In *Proceedings of the 2005 Conference on Uncertainty in Artificial Intelligence*, 201–208.

[Kaelbling, Littman, & Moore 1996] Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

[Konidaris & Barto 2006] Konidaris, G., and Barto, A. 2006. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, 489–496.

[Kuhlmann *et al.* 2004] Kuhlmann, G.; Stone, P.; Mooney, R.; and Shavlik, J. 2004. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*.

[Liu & Stone 2006] Liu, Y., and Stone, P. 2006. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 415–20.

[Maclin & Shavlik 1996] Maclin, R., and Shavlik, J. W. 1996. Creating advice-taking reinforcement learners. *Machine Learning* 22(1-3):251–281.

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

[McCarthy 1964] McCarthy, J. 1964. A tough nut for proof procedures. Technical Report Sail AI Memo 16, Computer Science Department, Stanford University.

[Milner 1982] Milner, R. 1982. *A Calculus of Communicating Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

[Noda *et al.* 1998] Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence* 12:233–250.

[Price & Boutilier 2003] Price, B., and Boutilier, C. 2003. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research* 19:569–629.

[Ravindran & Barto 2002] Ravindran, B., and Barto, A. 2002. Model minimization in hierarchical reinforcement learning. In *Proceedings of the Fifth Symposium on Abstraction, Reformulation and Approximation*.

[Ravindran & Barto 2003] Ravindran, B., and Barto, A. G. 2003. An algebraic approach to abstraction in reinforcement learning. In *Proceedings of the Twelfth Yale Workshop on Adaptive and Learning Systems*, 109–114.

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

[Şimşek & Barto 2006] Şimşek, Ö., and Barto, A. G. 2006. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning*.

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

[Skinner 1953] Skinner, B. F. 1953. *Science and Human Behavior*. Colliler-Macmillian.

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

[Stone *et al.* 2006] Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2006. Keepaway soccer: From machine learning testbed to benchmark. In Noda, I.; Jacoff, A.; Bredenfeld, A.; and Takahashi, Y., eds., *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020. Berlin: Springer-Verlag. 93–105.

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

[Sutton, Precup, & Singh 1999] Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

[Sutton 1988] Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–44.

[Szepesvári 2009] Szepesvári, C. 2009. Reinforcement learning algorithms for MDPs – a survey. Technical Report TR09-13, Dept. of Computing Science, University of Alberta.

[Taylor & Chernova 2010] Taylor, M. E., and Chernova, S. 2010. Integrating human demonstration and reinforcement learning: Initial results in human-agent transfer. In *Proceedings of the Agents Learning Interactively with Human Teachers AAMAS workshop*.

[Taylor & Stone 2007a] Taylor, M. E., and Stone, P. 2007a. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*.

[Taylor & Stone 2007b] Taylor, M. E., and Stone, P. 2007b. Representation transfer for reinforcement learning. In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*.

[Taylor & Stone 2009] Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(1):1633–1685.

[Taylor, Jong, & Stone 2008] Taylor, M. E.; Jong, N. K.; and Stone, P. 2008. Transferring instances for model-based reinforcement learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 488–505.

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

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

[Taylor 2009] Taylor, M. E. 2009. Assisting transfer-enabled machine learning algorithms: Leveraging human knowledge for curriculum design. In *The AAAI 2009 Spring Symposium on Agents that Learn from Human Teachers*.

[Thrun 1996] Thrun, S. 1996. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, volume 8, 640–646.

[Torrey *et al.* 2006] Torrey, L.; Shavlik, J. W.; Walker, T.; and Maclin, R. 2006. Skill acquisition via transfer learning and advice taking. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 425–436.

[Torrey *et al.* 2007] Torrey, L.; Shavlik, J. W.; Walker, T.; and Maclin, R. 2007. Relational macros for transfer in reinforcement learning. In *Proceedings of the Seventeenth Conference on Inductive Logic Programming*.

[Watkins 1989] Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College, Cambridge, UK.

[Widmer & Kubat 1996] Widmer, G., and Kubat, M. 1996. Learning in the presence of concept drift and hidden contexts. *Machine Learning* 23(1):69–101.

[Zang *et al.* 2010] Zang, P.; Irani, A.; Zhou, P.; Isbell, C.; and Thomaz, A. 2010. Combining function approximation, human teachers, and training regimens for real-world RL. In *The Ninth International Joint Conference on Autonomous Agents and Multiagent Systems*.