Journal of Machine Learning Research 10 (2009) 1633-1685 Submitted 6/08; Revised 5/09; Published 7/09

Transfer Learning for Reinforcement Learning Domains: A Survey

Matthew E. Taylor* TAYLORM@USC.EDU
Computer Science Department
The University of Southern California
Los Angeles, CA 90089-0781

Peter Stone PSTONE@CS.UTEXAS.EDU
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Editor: Sridhar Mahadevan

Abstract

The reinforcement learning paradigm is a popular way to address problems that have only limited environmental feedback, rather than correctly labeled examples, as is common in other machine learning contexts. While significant progress has been made to improve learning in a single task, the idea of *transfer learning* has only recently been applied to reinforcement learning tasks. The core idea of transfer is that experience gained in learning to perform one task can help improve learning performance in a related, but different, task. In this article we present a framework that classifies transfer learning methods in terms of their capabilities and goals, and then use it to survey the existing literature, as well as to suggest future directions for transfer learning work.

Keywords: transfer learning, reinforcement learning, multi-task learning

1. Transfer Learning Objectives

In *reinforcement learning* (RL) (Sutton and Barto, 1998) problems, leaning agents take sequential actions with the goal of maximizing a reward signal, which may be time-delayed. For example, an agent could learn to play a game by being told whether it wins or loses, but is never given the "correct" action at any given point in time. The RL framework has gained popularity as learning methods have been developed that are capable of handling increasingly complex problems. However, when RL agents begin learning *tabula rasa*, mastering difficult tasks is often slow or infeasible, and thus a significant amount of current RL research focuses on improving the speed of learning by exploiting domain expertise with varying amounts of human-provided knowledge. Common approaches include deconstructing the task into a hierarchy of subtasks (cf., Dietterich, 2000); learning with higher-level, temporally abstract, actions (e.g., *options*, Sutton et al. 1999) rather than simple one-step actions; and efficiently abstracting over the state space (e.g., via function approximation) so that the agent may generalize its experience more efficiently.

The insight behind *transfer learning* (TL) is that generalization may occur not only within tasks, but also *across tasks*. This insight is not new; transfer has long been studied in the psychological literature (cf., Thorndike and Woodworth, 1901; Skinner, 1953). More relevant are a number of

*. The first author wrote the majority of this article while a graduate student at the University of Texas at Austin.

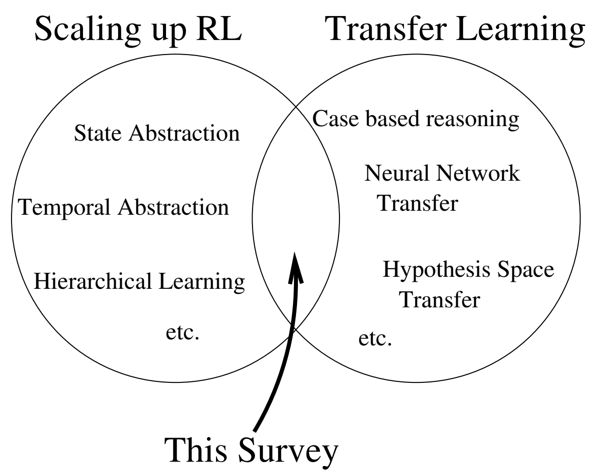

Figure 1: This article focuses on transfer between reinforcement learning tasks.

approaches that transfer between machine learning tasks (Caruana, 1995; Thrun, 1996), for planning tasks (Fern et al., 2004; Ilghami et al., 2005), and in the context of cognitive architectures (Laird et al., 1986; Choi et al., 2007). However, TL for RL tasks has only recently been gaining attention in the artificial intelligence community. Others have written surveys for reinforcement learning (Kaelbling et al., 1996), and for transfer across machine learning tasks (Thrun and Pratt, 1998), which we will not attempt to duplicate; this article instead focuses on transfer *between RL tasks* (see Figure 1) to provide an overview of a new, growing area of research.

Transfer learning in RL is an important topic to address at this time for three reasons. First, in recent years RL techniques have achieved notable successes in difficult tasks which other machine learning techniques are either unable or ill-equipped to address (e.g., TDGammon Tesauro 1994, job shop scheduling Zhang and Dietterich 1995, elevator control Crites and Barto 1996, helicopter control Ng et al. 2004, marble maze control Bentivegna et al. 2004, Robot Soccer Keepaway Stone et al. 2005, and quadruped locomotion Saggar et al. 2007 and Kolter et al. 2008). Second, classical machine learning techniques such as rule induction and classification are sufficiently mature that they may now easily be leveraged to assist with TL. Third, promising initial results show that not only are such transfer methods possible, but they can be very effective at speeding up learning. The 2005 DARPA Transfer Learning program (DARPA, 2005) helped increase interest in transfer learning. There have also been some recent workshops providing exposure for RL techniques that use transfer. The 2005 NIPS workshop, “Inductive Transfer: 10 Years Later,” (Silver et al., 2005) had few RL-related transfer papers, the 2006 ICML workshop, “Structural Knowledge Transfer for Machine Learning,” (Banerjee et al., 2006) had many, and the 2008 AAAI workshop, “Transfer Learning for Complex Tasks,” (Taylor et al., 2008a) focused on RL.

1.1 Paper Overview

The goals of this survey are to introduce the reader to the transfer learning problem in RL domains, to organize and discuss current transfer methods, and to enumerate important open questions in RL transfer. In transfer, knowledge from one or more *source task(s)* is used to learn one or more *target task(s)* faster than if transfer was not used. The literature surveyed is structured primarily by grouping methods according to how they allow source and target tasks to differ. We further distinguish methods according to five different dimensions (see Section 2.2). Some of the questions that distinguish transfer methods include:

- What are the goals of the transfer method? By what metric(s) will success be measured? Section 2 examines commonly used metrics, as well as different settings where transfer learning can improve learning.
- What assumptions, if any, are made regarding the similarity between the tasks? Section 3.2.1 enumerates common differences, such as changes to the space in which agents operate, allowing the agents to have different goals, or letting agents have different sets of actions.
- How does a transfer method identify what information can/should be transferable? Section 3.2.2 enumerates possibilities ranging from assuming *all* previously seen tasks are directly useful to autonomously learning which source task(s) are useful for learning in the current target task.
- What information is transferred between tasks? Section 3.2.3 discusses possibilities ranging from very low-level information (such as direct control knowledge) to high-level information (such as rules regarding how a particular domain functions).

The following section presents a discussion about how to best evaluate transfer in RL. There are many different situations in which transfer can be useful and these different situations may entail different metrics. This discussion will prepare the reader to better understand how transfer may be used. Section 3.1 will briefly discuss reinforcement learning and the notation used in the article. Section 3.2 enumerates the ways in which transfer methods can differ, providing a skeleton for the structure of this survey. Sections 3.3 and 3.4 provide additional high-level categorization of TL methods and Section 3.5 discusses related learning paradigms which are explicitly not discussed in this survey.

The bulk of the remainder of the article (Sections 4–8) discuss contemporary TL methods, arranged by the goals of, and methods employed by, the designers. Lastly, Section 9 discusses current open questions in transfer and concludes.

2. Evaluating Transfer Learning Methods

Transfer techniques assume varying degrees of autonomy and make many different assumptions. 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.

A key challenge in TL research is to define evaluation metrics, precisely because there are many possible measurement options and algorithms may focus on any of the three steps above. This section focuses on how to best evaluate TL algorithms so that the reader may better understand the different goals of transfer and the situations where transfer may be beneficial.[1] 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 as a "sunk cost." On the one hand, a 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 or tasks, would be most appropriate. On the other hand, a second reasonable goal of transfer is to effectively reuse past knowledge in a novel task. In this case, a *target task time scenario*, which only accounts for the time spent learning in the target task, is reasonable.

The total time scenario may be more appropriate when an agent is explicitly guided by a human. Suppose that a user wants an agent to learn to perform a task, but recognizes that the agent may be able to learn a sequence of tasks faster than if it directly tackled the difficult task. The human can construct a series of tasks for the agent, suggesting to the agent how the tasks are related. Thus the agent's TL method will easily accomplish steps 1 and 2 above, but it must efficiently transfer knowledge between tasks (step 3). To successfully transfer in this setting, the agent would have to learn the entire sequence of tasks faster than if it had spent its time learning the final target task directly (see the total time scenario in Figure 2).

The target task time scenario is more appropriate for a fully autonomous learner. A fully autonomous agent must be able to perform steps 1–3 on its own. However, metrics for this scenario do not need to take into account the cost of learning source tasks. The target task time scenario emphasizes the agent's ability to use knowledge from one or more previously learned source tasks without being charged for the time spent learning them (see the target task time scenario in Figure 2). In this survey we will see that the majority of existing transfer algorithms assume a human-guided scenario, but disregard time spent training in the source task. When discussing individual TL methods, we will specifically call attention to the methods that do account for the total training time and do not treat the time spent learning a source task as a sunk cost.

Many metrics to measure the benefits of transfer are possible (shown in Figure 3, replicated from our past transfer learning work, Taylor and Stone 2007b):

1. *Jumpstart*: The initial performance of an agent in a target task may be improved by transfer from a source task.

2. *Asymptotic Performance*: The final learned performance of an agent in the target task may be improved via transfer.

3. *Total Reward*: The total reward accumulated by an agent (i.e., the area under the learning curve) may be improved if it uses transfer, compared to learning without transfer.

4. *Transfer Ratio*: The ratio of the total reward accumulated by the transfer learner and the total reward accumulated by the non-transfer learner.

5. *Time to Threshold*: The learning time needed by the agent to achieve a pre-specified performance level may be reduced via knowledge transfer.

Metrics 1–4 are most appropriate in the fully autonomous scenario as they do not charge the agent for time spent learning any source tasks. To measure the total time, the metric must account for time

1. Evaluation is particularly important because there are very few theoretical results supporting TL for RL methods, as discussed further in Section 9.3. Instead, practitioners rely on empirical methods to evaluate the efficacy of their methods.

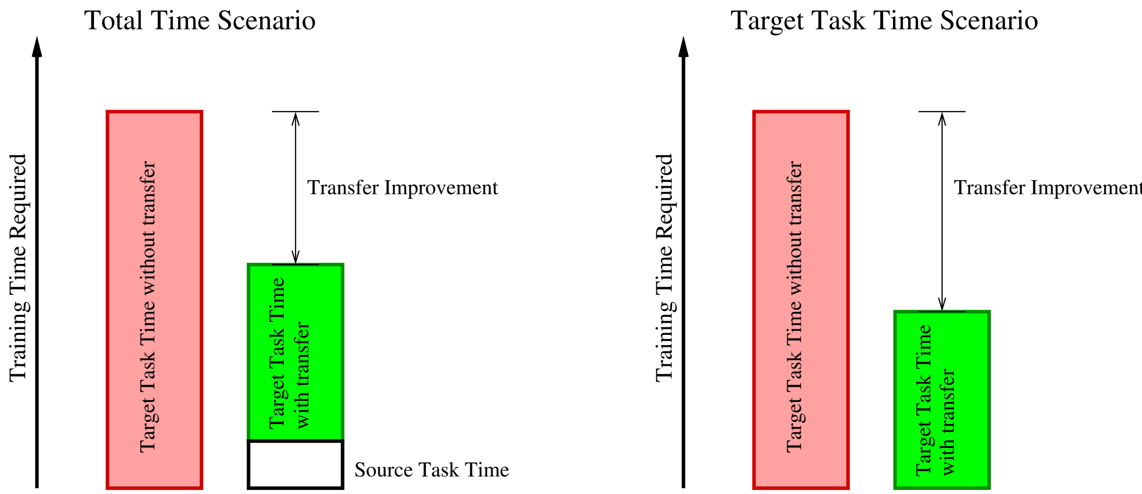

Figure 2: Successful TL methods may be able to reduce the total training time (left). In some scenarios, it is more appropriate to treat the source task time as a sunk cost and test whether the method can effectively reuse past knowledge to reduce the target task time (right).

spent learning one or more source tasks, which is natural when using metric 5. Other metrics have been proposed in the literature, but we choose to focus on these five because they are sufficient to describe the methods surveyed in this article.

For this article, we may think of learning time as a surrogate for *sample complexity*. Sample complexity (or data complexity) in RL refers to the amount of data required by an algorithm to learn. It is strongly correlated with learning time because RL agents only gain data by collecting it through repeated interactions with an environment.

2.1 Empirical Transfer Comparisons

The previous section enumerated five possible TL metrics, and while others are possible, these represent the methods most commonly used. However, each metric has drawbacks and none are sufficient to fully describe the benefits of any transfer method. Rather than attempting to create a total order ranking of different methods, which may indeed by impossible, we instead suggest that a multi-dimensional evaluation with multiple metrics is most useful. Specifically, some methods may "win" on a set of metrics relative to other methods, but "lose" on a different set. As the field better understands why different methods achieve different levels of success on different metrics, it should become easier to map TL methods appropriately to TL problems. Although the machine learning community has defined standard metrics (such as precision vs. recall curves for classification and mean squared error for regression), RL has no such standard. Empirically comparing two RL algorithms is a current topic of debate within the community, although there is some process towards standardizing comparisons (Whiteson et al., 2008). Theoretical comparisons are also not clear-cut, as samples to convergence, asymptotic performance, and the computational complexity are all valid axes along which to evaluate RL algorithms.

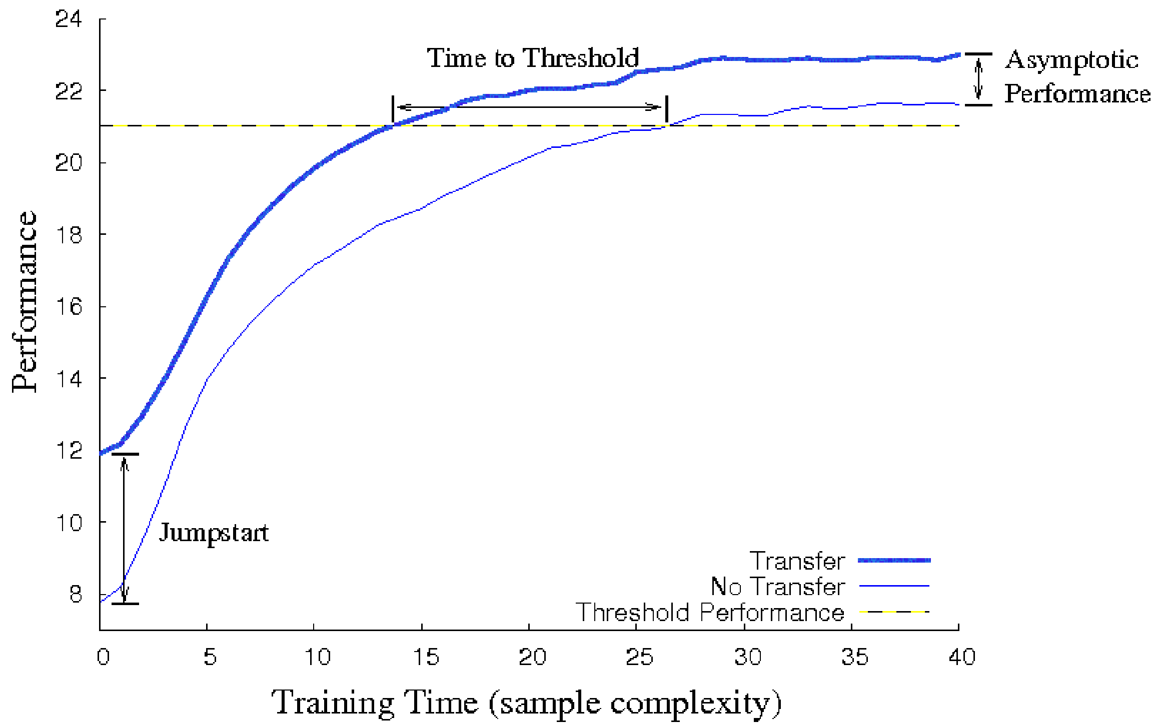

Figure 3: Many different metrics for measuring TL are possible. This graph show benefits to the jumpstart, asymptotic performance, time to threshold, and total reward (the area under the learning curve).

The first proposed transfer measure considers the agent's initial performance in a target task and answers the question, "can transfer be used so that the initial performance is increased relative to the performance of an initial (random) policy?" While such an initial jumpstart is appealing, such a metric fails to capture the behavior of *learning* in the target task and instead only focuses on the performance before learning occurs.

Asymptotic performance, the second proposed metric, compares the final performance of learners in the target task both with and without transfer. However, it may be difficult to tell when the learner has indeed converged (particularly in tasks with infinite state spaces) or convergence may take prohibitively long. In many settings the number of samples required to learn is most critical, not the performance of a learner with an infinite number of samples. Further, it is possible for different learning algorithms to converge to the same asymptotic performance but require very different numbers of samples to reach the same performance.

A third possible measure is that of the total reward accumulated during training. Improving initial performance and achieving a faster learning rate will help agents accumulate more on-line reward. RL methods are often not guaranteed to converge with function approximation and even when they do, learners may converge to different, sub-optimal performance levels. If enough samples are provided to agents (or, equivalently, learners are provided sufficient training time), a learning method which achieves a high performance relatively quickly will have less total reward than a learning method which learns very slowly but eventually plateaus at a slightly higher performance level. This metric is most appropriate for tasks that have a well-defined duration.

A fourth measure of transfer efficacy is that of the ratio of areas defined by two learning curves. Consider two learning curves in the target task where one uses transfer and one does not. Assuming that the transfer learner accrues more reward, the area under the transfer leaning curve will be greater than the area under the non-transfer learning curve. The ratio

$$r = \frac{\textit{area under curve with transfer - area under curve without transfer}}{\textit{area under curve without transfer}}$$

gives a metric that quantifies improvement from TL. This metric is most appropriate if the same final performance is achieved, or there is a predetermined time for the task. Otherwise the ratio will directly depend on how long the agents act in the target task.

While such a metric may be appealing as a candidate for inter-task comparisons, we note that the transfer ratio is not scale invariant. For instance, if the area under the transfer curve were 1000 units and the area under the non-transfer curve were 500, the transfer ratio would be 1.0. If all rewards were multiplied by a constant, this ratio would not change. But if an offset were added (e.g., each agent is given an extra +1 at the end of each episode, regardless of the final state), the ratio would change. The evaluation of a TL algorithm with the transfer ratio is therefore closely related to the reward structure of the target task being tested. Lastly, we note that although none of the papers surveyed in this article use such a metric, we hope that it will be used more often in the future.

The final metric, Time to Threshold, suffers from having to specify a (potentially arbitrary) performance agents must achieve. While there have been some suggestions how to pick such thresholds appropriately (Taylor et al., 2007a), the relative benefit of TL methods will clearly depend on the exact threshold chosen, which will necessarily be domain- and learning method-dependent. While choosing a range of thresholds to compare over may produce more representative measures (cf., Taylor et al., 2007b), this leads to having to generating a time vs. threshold curve rather than producing a single real valued number that evaluates a transfer algorithm's efficacy.

A further level of analysis that could be combined with any of the above methods would be to calculate a ratio comparing the performance of a TL algorithm with that of a human learner. For instance, a set of human subjects could learn a given target task with and without having first trained on a source task. By averaging over their performances, different human transfer metrics could be calculated and compared to that of a TL algorithm. However, there are many ways to manipulate such a meta-metric. For instance, if a target task is chosen that humans are relatively proficient at, transfer will provide them very little benefit. If that same target task is difficult for a machine learning algorithm, it will be relatively easy to show that the TL algorithm is quite effective relative to human transfer, even if the agent's absolute performance is extremely poor.

A major drawback of all the metrics discussed is that none are appropriate for inter-domain comparisons. The vast majority of papers in this survey compare learning with and without transfer—their authors often do not attempt to directly compare different transfer methods. Developing fair metrics that apply across multiple problem domains would facilitate better comparisons of methods. Such inter-domain metrics may be infeasible in practice, in which case standardizing on a set of test domains would assist in comparing different TL methods (as discussed further in Section 9). In the absence of either a set of inter-domain metrics or a standard benchmark suite of domains, we limit our comparisons of different TL methods in this survey to their applicability, assumptions, and algorithmic differences. When discussing different methods, we may opine on the method's relative performance, but we remind the reader that such commentary is largely based on intuition rather than empirical data.

2.2 Dimensions of Comparison

In addition to differing on evaluation metrics, we categorize TL algorithms along five dimensions, which we use as the main organizing framework for our survey of the literature:

I *Task difference assumptions*: What assumptions does the TL method make about how the source and target are allowed to differ? Examples of things that can differ between the source and target tasks include different system dynamics (i.e., the target task becomes harder to solve is some incremental way), or different sets of possible actions at some states. Such assumptions define the types of source and target tasks that the method can transfer between. Allowing transfer to occur between less similar source and target tasks gives more flexibility to a human designer in the human-guided scenario. In the fully autonomous scenario, more flexible methods are more likely to be able to successfully apply past knowledge to novel target tasks.

II *Source task selection*: In the simplest case, the agent assumes that a human has performed source task selection (the human-guided scenario), and transfers from one or more selected tasks. More complex methods allow the agent to select a source task or set of source tasks. Such a selection mechanism may additionally be designed to guard against *negative transfer*, where transfer hurts the learner's performance. The more robust the selection mechanism, the more likely it is that transfer will be able to provide a benefit. While no definitive answer to this problem exists, successful techniques will likely have to account for specific target task characteristics. For instance, Carroll and Seppi (2005) motivate the need for general task similarity metrics to enable robust transfer, propose three different metrics, and then proceed to demonstrate that none is always "best," just as there is never a "best" inductive bias in a learning algorithm.

III *Task Mappings*: Many methods require a mapping to transfer effectively: in addition to knowing that a source task and target task are related, they need to know *how* they are related. *Inter-task mappings* (discussed in detail later in Section 3.4) are a way to define how two tasks are related. If a human is in the loop, the method may assume that such task mappings are provided; if the agent is expected to transfer autonomously, such mappings have to be learned. Different methods use a variety of techniques to enable transfer, both on-line (while learning the target task) and offline (after learning the source task but before learning the target task). Such learning methods attempt to minimize the number of samples needed and/or the computational complexity of the learning method, while still learning a mapping to enable effective transfer.

IV *Transferred Knowledge*: What type of information is transferred between the source and target tasks? This information can range from very low-level information about a specific task (i.e., the expected outcome when performing an action in a particular location) to general heuristics that attempt to guide learning. Different types of knowledge may transfer better or worse depending on task similarity. For instance, low-level information may transfer across closely related tasks, while high-level concepts may transfer across pairs of less similar tasks. The mechanism that transfers knowledge from one task to another is closely related to what is being transferred, how the task mappings are defined (III), and what assumptions about the two tasks are made (I).

V *Allowed Learners*: Does the TL method place restrictions on what RL algorithm is used, such as applying only to temporal difference methods? Different learning algorithms have different biases. Ideally an experimenter or agent would select the RL algorithm to use based on characteristics of the task, not on the TL algorithm. Some TL methods require that the source and target tasks be learned with the same method, other allow a class of methods to be used in both tasks, but the most flexible methods decouple the agents' learning algorithms in the two tasks.

An alternate TL framework may be found in the related work section of Lazaric (2008), a recent PhD thesis on TL in RL tasks. Lazaric compares TL methods in terms of the type of benefit (jumpstart, total reward, and asymptotic performance), the allowed differences between source and target (different goal states, different transition functions but the same reward function, and different state and action spaces) and the type of transferred knowledge (experience or structural knowledge). Our article is more detailed both in the number of approaches considered, the depth of description about each approach, and also uses a different organizational structure. In particular, we specify which of the methods improve which of five TL metrics, we note which of the methods account for source task training time rather than treating it as a sunk cost, and we differentiate methods according to five dimensions above.

3. Transfer for Reinforcement Learning

In this section we first give a brief overview of notation. We then summarize the methods discussed in this survey using the five dimensions previously discussed, as well as enumerating the possible attributes for these dimensions. Lastly, learning paradigms with goals similar to transfer are discussed in Section 3.5.

3.1 Reinforcement Learning Background

RL problems are typically framed in terms of *Markov decision processes* (MDPs) (Puterman, 1994). For the purposes of this article, *MDP* and *task* are used interchangeably. In an MDP, there is some set of possible perceptions of the current *state* of the world, $s \in S$, and a learning agent has one or more initial starting states, $s_{initial}$. The *reward function*, $R : S \mapsto \mathbb{R}$, maps each state of the environment to a single number which is the instantaneous reward achieved for reaching the state. If the task is *episodic*, the agent begins at a start state and executes actions in the environment until it reaches a terminal state (one or more of the states in s_{final}, which may be referred to as a *goal state*), at which point the agent is returned to a start state. An agent in an episodic task typically attempts to maximize the average reward per episode. In non-episodic tasks, the agent attempts to maximize the total reward, which may be discounted. By using a discount factor, γ, the agent can weigh immediate rewards more heavily than future rewards, allowing it to maximize a non-infinite sum of rewards.

An agent knows its current state in the environment, $s \in S$.[2] TL methods are particularly relevant in MDPs that have a large or continuous state, as these are the problems which are slow to learn *tabula rasa* and for which transfer may provide substantial benefits. Such tasks typically factor the state using *state variables* (or *features*), so that $s = \langle x_1, x_2, \ldots, x_n \rangle$ (see Figure 4). The agent's observed state may be different from the true state if there is perceptual noise. The set A describes the *actions* available to the agent, although not every action may be possible in every state.[3] 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. A learner 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 a learner 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. An *optimal policy*, π^*, is a policy which does maximize the expectation of this value. Any reasonable learning algorithm attempts to modify π over time so that the agent's performance approaches that of π^* in the limit.

There are many possible approaches to learning such a policy (depicted as a black box in Figure 4), including:

- *Temporal difference* (TD) methods, such as *Q-learning* (Sutton, 1988; Watkins, 1989) and *Sarsa* (Rummery and Niranjan, 1994; Singh and Sutton, 1996), learn by backing up experienced rewards through time. An estimated *action-value function*, $Q : S \times A \mapsto \mathbb{R}$ is learned, where $Q(s,a)$ is the expected return found when executing action a from state s, and greedily following the current policy thereafter. The current best policy is generated from Q by simply selecting the action that has the highest value for the current state. *Exploration*, when the agent chooses an action to learn more about the environment, must be balanced with *exploitation*, when the agent selects what it believes to be the best action. One simple approach that balances the two is ϵ-greedy action selection: the agent selects an random action with chance ϵ, and the current best action is selected with probability $1-\epsilon$ (where ϵ is in [0,1]).

- *Policy search* methods, such as policy iteration (dynamic programming), policy gradient (Williams, 1992; Baxter and Bartlett, 2001), and direct policy search (Ng and Jordan, 2000), are in some sense simpler than TD methods because they directly modify a policy over time to increase the expected long-term reward by using search or other optimization techniques.

- *Dynamic programming* (Bellman, 1957) approaches assume that a full model of the environment is known (i.e., S, A, T, and R are provided to the agent and are correct). No interaction with the environment is necessary, but the agent must iteratively compute approximations for the true value or action-value function, improving them over time.

- *Model-based* or *Model-learning* methods (Moore and Atkeson, 1993; Kearns and Singh, 1998) attempt to estimate the true model of the environment (i.e., T and R) by interacting

2. If the agent only receives *observations* and does not know the true state, the agent may treat approximate its true state as the observation (cf., Stone et al., 2005), or it may learn using the Partially Observable Markov Decision Process (POMDP) (cf., Kaelbling et al., 1998) problem formulation, which is beyond the scope of this survey.
3. Although possible in principle, we are aware of no TL methods currently address MDPs with continuous actions.

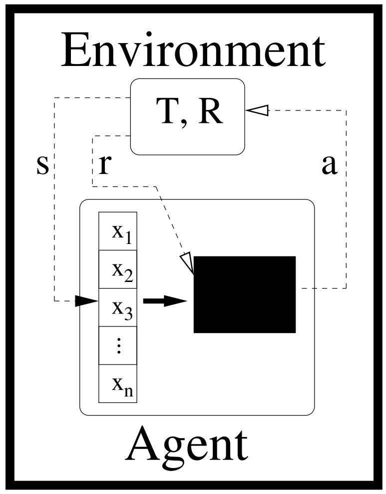

Figure 4: An agent interacts with an environment by sequentially selecting an action in an observed state, with the objective of maximizing an environmental reward signal.

with the environment over time. *Instance based methods* (Ormoneit and Sen, 2002) save observed interactions with the environment and leverage the instance directly to predict the model. *Bayesian RL* (Dearden et al., 1999) approaches use a mathematical model to explicitly represent uncertainty in the components of the model, updating expectations over time. The learned model is then typically used to help the agent decide how to efficiently explore or plan trajectories so that it can accrue higher rewards. While very successful in small tasks, few such methods handle continuous state spaces (cf., Jong and Stone, 2007), and they generally have trouble scaling to tasks with many state variables due to the "curse of dimensionality."

- *Relational reinforcement learning* (RRL) (Dzeroski et al., 2001) uses a different learning algorithm as well as a different state representation. RRL may be appropriate if the state of an MDP can be described in a relational or first-order language. Such methods work by reasoning over individual objects (e.g., a single block in a Blocksworld task) and thus may be robust to changes in numbers of objects in a task.

- *Batch* learning methods (e.g., *Least Squares Policy Iteration* (Lagoudakis and Parr, 2003) and Fitted-Q Iteration (Ernst et al., 2005) are offline and do not attempt to learn as the agent interacts with the environment. Batch methods are designed to be more sample efficient, as they can store a number of interactions with the environment and use the data multiple times for learning. Additionally, such methods allow a clear separation of the learning mechanism from the exploration mechanism (which much decide whether to attempt to gather more data about the environment or exploit the current best policy).

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. In such cases, RL learning methods use *function approximators*, such as artificial neural networks, which rely on concise, parameterized functions and use supervised learning methods to set these parameters. Function approximation is used in large or continuous tasks to better generalize experience. Parameters and biases in the approximator are used to abstract the state space so that observed data can influence a region of state space, rather than just a single state, and can substantially increase the speed of learning.

Some work in RL (Dean and Givan, 1997; Li et al., 2006; Mahadevan and Maggioni, 2007) has experimented with more systematic approaches to state abstractions (also called structural abstraction). Temporal abstractions have also been successfully used to increase the speed of learning. These macro-actions or *options* (Sutton et al., 1999) may allow the agent to leverage the sequence of actions to learn its task with less data. Lastly, hierarchical methods, such as MAXQ (Dietterich, 2000), allow learners exploit a task that is decomposed into different sub-tasks. The decomposition typically enables an agent to learn each subtask relatively quickly and then combine them, resulting in an overall learning speed improvement (compared to methods that do not leverage such a sub-task hierarchy).

3.2 Transfer Approaches

Having provided a brief overview of the RL notation used in this survey, we now enumerate possible approaches for transfer between RL tasks. This section lists attributes of methods used in the TL literature for each of the five dimensions discussed in Section 2.2, and summarizes the surveyed works in Table 1. The first two groups of methods apply to tasks which have the same state variables and actions. (Section 4 discusses the TL methods in the first block, and Section 5 discusses the multi-task methods in the second block.) Groups three and four consider methods that transfer between tasks with different state variables and actions. (Section 6 discusses methods that use a representation that does not change when the underlying MDP changes, while Section 7 presents methods that must explicitly account for such changes.) The last group of methods (discussed in Section 8) learns a mapping between tasks like those used by methods in the fourth group of methods. Table 2 concisely enumerates the possible values for the attributes, as well as providing a key to Table 1.

In this section the *mountain car* task (Moore, 1991; Singh and Sutton, 1996), a standard RL benchmark, will serve as a running example. In mountain car, an under-powered car moves along a curve and attempts to reach a goal state at the top of the right "mountain" by selecting between three actions on every timestep: {`Forward`, `Neutral`, `Backward`}, where `Forward` accelerates the car in the positive x direction and `Backward` accelerates the car in the negative x direction. The agent's state is described by two state variables: the horizontal position, x, and velocity, $\dot{x}$. The agent receives a reward of -1 on each time step. If the agent reaches the goal state the episode ends and the agent is reset to the start state (often the bottom of the hill, with zero velocity).

3.2.1 ALLOWED TASK DIFFERENCES

TL methods can transfer between MDPs that have different transition functions (denoted by t in Table 1), state spaces (s), start states (s_i), goal states (s_f), state variables (v), reward functions (r), and/or action sets (a). For two of the methods, the agent's representation of the world (the *agent-space*, describing physical sensors and actuators) remains the same, while the true state variables and actions (the *problem-space*, describing the task's state variables and macro-actions) can change (p in Table 1, discussed further in Section 6). There is also a branch of work that focuses on transfer between tasks which are composed of some number of objects that may change between the source and the target task, such as when learning with RRL (# in Table 1). When summarizing the allowed task differences, we will concentrate on the most salient features. For instance, when the source task and target task are allowed to have different state variables and actions, the state space of the two tasks is different because the states are described differently, and the transition function and reward function must also change, but we only indicate "a" and "v."

These differences in the example mountain car task could be exhibited as:

- t: using a more powerful car motor or changing the surface friction of the hill
- s: changing the range of the state variables
- s_i: changing where the car starts each episode
- s_f: changing the goal state of the car
- v: describing the agent's state only by its velocity
- r: rather than a reward of -1 on every step, the reward could be a function of the distance from the goal state
- a: disabling the `Neutral` action
- p: the agent could describe the state by using extra state variables, such as the velocity on the previous timestep, but the agent only directly measures its current position and velocity
- #: the agent may need to control two cars simultaneously on the hill

3.2.2 SOURCE TASK SELECTION

The simplest method for selecting a source task for a given target task is to assume that only a single source task has been learned and that a human has picked it, assuring that the agent should use it for transfer (h in Table 1). Some TL algorithms allow the agent to learn multiple source tasks and then use them all for transfer (all). More sophisticated algorithms build a library of seen tasks and use only the most relevant for transfer (lib). Some methods are able to automatically modify a single source task so that the knowledge it gains from the modified task will likely be more useful in the target task (mod). However, none of the existing TL algorithms for RL can guarantee that the source tasks will be useful; a current open question is how to robustly avoid attempting to transfer from an irrelevant task.

3.2.3 TRANSFERRED KNOWLEDGE

The type of knowledge transferred can be primarily characterized by its specificity. Low-level knowledge, such as $\langle$ s, a, r, s$'$ $\rangle$ instances (I in Table 1), an action-value function (Q), a policy (π), a full task model (model), or prior distributions (pri), could all be directly leveraged by the TL algorithm to initialize a learner in the target task. Higher level knowledge, such as what action to use in some situations (A: a subset of the full set of actions), partial policies or options (π_p), rules or advice (rule), important features for learning (fea), proto-value functions (pvf: a type of learned feature), shaping rewards (R), or subtask definitions (sub) may not be directly used by the algorithm to fully define an initial policy, but such information may help guide the agent during learning in the target task.

3.2.4 Task Mappings

The majority of TL algorithms in this survey assume that no explicit task mappings are necessary because the source and target task have the same state variables and actions. In addition to having the same labels, the state variables and actions need to have the same semantic meanings in both tasks. For instance, consider again the mountain car domain. Suppose that the source task had the actions A = {`Forward, Neutral, Backward`}. If the target task had the actions A = {`Right, Neutral, Left`}, a TL method would need some kind of mapping because the actions had different labels. Furthermore, suppose that the target task had the same actions as the source (A = {`Forward, Neutral, Backward`}) but the car was facing the opposite direction, so that `Forward` accelerated the car in the negative `x` direction and `Backward` accelerated the car in the positive `x` direction. If the source and target task actions have different semantic meanings, there will also need to be some kind of inter-task mapping to enable transfer.

Methods that do not use a task mapping are marked as "N/A" in Table 1. TL methods which aim to transfer between tasks with different state variables or actions typically rely on a task mapping to define how the tasks are related (as defined in Section 3.4). Methods that use mappings and assume that they are human-supplied mappings are marked as "sup" in Table 1. A few algorithms leverage experience gained in the source task and target task (exp) or a high-level description of the MDPs in order to learn task mappings.

Methods using description-level knowledge differ primarily in what assumptions they make about what will be provided. One method assumes a qualitative understanding of the transition function (T), which would correspond to knowledge like "taking the action Neutral tends to have a positive influence on the velocity in the positive x direction." Two methods assume knowledge of one mapping (M_a: the "action mapping") to learn a second mapping (the "state variable mapping" in Section 3.4). Three methods assume that the state variables are "grouped" together to describe objects (sv_g). An example of the state variable grouping can be demonstrated in a mountain car task with multiple cars: if the agent knew which position state variables referred to the same car as certain velocity state variables, it would know something about the grouping of state variables. These different assumptions are discussed in detail in Section 8.

3.2.5 Allowed Learners

The type of knowledge transferred directly affects the type of learner that is applicable (as discussed in Section 3.1). For instance, a TL method that transfers an action-value function would likely require that the target task agent use a temporal difference method to exploit the transferred knowledge. The majority of methods in the literature use a standard form of temporal difference learning (TD in Table 1), such as Sarsa. Other methods include Bayesian learning (B), hierarchical approaches (H), model-based learning (MB), direct policy search (PS), and relational reinforcement learning (RRL). Some TL methods focus on batch learning (Batch), rather than on-line learning. Two methods use *case based reasoning* (CBR) (Aamodt and Plaza, 1994) to help match previously learned instances with new instances, and one uses linear programming (LP) to calculate a value function from a given model (as part of a dynamic programming routine).

3.3 Multi-Task Learning

Closely related to TL algorithms, and discussed in Section 5, are *multi-task learning* (MTL) algorithms. The primary distinction between MTL and TL is that multi-task learning methods assume

Citation	Allowed Task Differences	Source Task Selection	Task Mappings	Transferred Knowledge	Allowed Learners	TL Metrics
Same state variables and actions: Section 4						
Selfridge et al. (1985)	t	h	N/A	Q	TD	tt†
Asada et al. (1994)	s_i	h	N/A	Q	TD	tt
Singh (1992)	r	all	N/A	Q	TD	ap, tr
Atkeson and Santamaria (1997)	r	all	N/A	model	MB	ap, j, tr
Asadi and Huber (2007)	r	h	N/A	π_p	H	tt
Andre and Russell (2002)	r, s	h	N/A	π_p	H	tr
Ravindran and Barto (2003b)	s, t	h	N/A	π_p	TD	tr
Ferguson and Mahadevan (2006)	r, s	h	N/A	pvf	Batch	tt
Sherstov and Stone (2005)	s_f, t	mod	N/A	A	TD	tr
Madden and Howley (2004)	s, t	all	N/A	rule	TD	tt, tr
Lazaric (2008)	s, t	lib	N/A	I	Batch	j, tr
Multi-Task learning: Section 5						
Mehta et al. (2008)	r	lib	N/A	π_p	H	tr
Perkins and Precup (1999)	t	all	N/A	π_p	TD	tt
Foster and Dayan (2004)	s_f	all	N/A	sub	TD, H	j, tr
Fernandez and Veloso (2006)	s_i, s_f	lib	N/A	π	TD	tr
Tanaka and Yamamura (2003)	t	all	N/A	Q	TD	j, tr
Sunmola and Wyatt (2006)	t	all	N/A	pri	B	j, tr
Wilson et al. (2007)	r, s_f	all	N/A	pri	B	j, tr
Walsh et al. (2006)	r, s	all	N/A	fea	any	tt
Lazaric (2008)*	r	all	N/A	fea	Batch	ap, tr
Different state variables and actions – no explicit task mappings: Section 6						
Konidaris and Barto (2006)	p	h	N/A	R	TD	j, tr
Konidaris and Barto (2007)	p	h	N/A	π_p	TD	j, tr
Banerjee and Stone (2007)	a, v	h	N/A	fea	TD	ap, j, tr
Guestrin et al. (2003)	#	h	N/A	Q	LP	j
Croonenborghs et al. (2007)	#	h	N/A	π_p	RRL	ap, j, tr
Ramon et al. (2007)	#	h	N/A	Q	RRL	ap, j, tt†, tr
Sharma et al. (2007)	#	h	N/A	Q	TD, CBR	j, tr
Different state variables and actions – inter-task mappings used: Section 7						
Taylor et al. (2007a)	a, v	h	sup	Q	TD	tt†
Taylor et al. (2007b)	a, v	h	sup	π	PS	tt†
Taylor et al. (2008b)	a, v	h	sup	I	MB	ap, tr
Torrey et al. (2005) Torrey et al. (2006)	a, r, v	h	sup	rule	TD	j, tr
Torrey et al. (2007)	a, r, v	h	sup	π_p	TD	j, tr
Taylor and Stone (2007b)	a, r, v	h	sup	rule	any/TD	j, tt†, tr
Learning inter-task mappings: Section 8						
Kuhlmann and Stone (2007)	a, v	h	T	Q	TD	j, tr
Liu and Stone (2006)	a, v	h	T	N/A	all	N/A
Soni and Singh (2006)	a, v	h	M_a, sv_g, exp	N/A	all	ap, j, tr
Talvitie and Singh (2007)	a, v	h	M_a, sv_g, exp	N/A	all	j
Taylor et al. (2007b)*	a, v	h	sv_g, exp	N/A	all	tt†
Taylor et al. (2008c)	a, v	h	exp	N/A	all	j, tr

Table 1: This table lists all the TL methods discussed in this survey and classifies each in terms of the five transfer dimensions (the key for abbreviations is in Table 2). Two entries, marked with a $\star$, are repeated due to multiple contributions. Metrics that account for source task learning time, rather than ignoring it, are marked with a †.

	Allowed Task Differences		Transferred Knowledge
a	action set may differ	A	an action set
p	problem-space may differ	fea	task features
	(agent-space must be identical)	I	experience instances
r	reward function may differ	model	task model
s_i	the start state may change	π	policies
s_f	goal state may move	π_p	partial policies (e.g., options)
t	transition function may differ	pri	distribution priors
v	state variables may differ	pvf	proto-value function
#	number of objects in state may differ	Q	action-value function
		R	shaping reward
		rule	rules or advice
		sub	subtask definitions
	Source Task Selection		
all	all previously seen tasks are used		Allowed Learners
h	one source task is used (human selected)	B	Bayesian learner
lib	tasks are organized into a library	Batch	batch learner
	and one or more may be used	CBR	case based reasoning
mod	a human provides a source task that	H	hierarchical value-function learner
	the agent automatically modifies	LP	linear programming
		MB	model based learner
	Task Mappings	PS	policy search learner
exp	agent learns the mappings from experience	RRL	relational reinforcement learning
M_a	the method must be provided with an	TD	temporal difference learner
	action mapping (learns state variable mapping)		
N/A	no mapping is used		TL Metrics
sup	a human supplies the task mappings	ap	asymptotic performance increased
sv_g	method is provided groupings of state variables	j	jumpstart demonstrated
T	higher-level knowledge is provided	tr	total reward increased
	about transfer functions to learn mapping	tt	task learning time reduced

Table 2: This key provides a reference to the abbreviations in Table 1.

all problems experienced by the agent are drawn from the same distribution, while TL methods may allow for arbitrary source and target tasks. For example, a MTL task could be to learn a series of mountain car tasks, each of which had a transition function that was drawn from a fixed distribution of functions that specified a range of surface frictions. Because of this assumption, MTL methods generally do not need task mappings (dimension III in Section 2.2). MTL algorithms may be used to transfer knowledge between learners, similar to TL algorithms, or they can attempt to learn how to act on the entire class of problems.

When discussing supervised multitask learning (cf., Caruana, 1995, 1997), data from multiple tasks can be considered simultaneously. In an RL setting, rather than trying to learn multiple problems simultaneously (i.e., acting in multiple MDPs), agents tackle a sequence of tasks which are more closely related than in TL settings. It is possible that RL agents could learn multiple tasks simultaneously in a multiagent setting (Stone and Veloso, 2000), but this has not yet been explored in the literature. For the purposes of this survey, we will assume, as in other transfer settings, that tasks are learned in a sequential order.

Sutton et al. (2007) motivate this approach to transfer by suggesting that a single large task may be most appropriately tackled as a sequential series of subtasks. If the learner can track which subtask it is currently in, it may be able to transfer knowledge between the different subtasks, which are all presumably related because they are part of the same overall task. Such a setting may provide a well-grounded way of selecting a distribution of tasks to train over, either in the context of transfer or for multi-task learning. Note also that the additional assumptions in an MTL setting may be leveraged to allow a more rigorous theoretical analysis than in TL (cf., Kalmár and Szepesvári, 1999).

3.4 Inter-Task Mappings

Transfer methods that assume the source and target tasks use the same state variables and actions, as is the case in MTL, typically do not need an explicit mapping between task. 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 et al., 2007a), one formulation of task mappings. Task mappings like these are used by transfer methods discussed in Section 7.

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. For instance, if the agent learns to act in a source task with the actions `Forward` and `Backward`, but the target task uses the actions `Right` and `Left`, the correspondence between these action sets may not be obvious. Even if the action labels were the same, if the actions had different semantic meanings, the default correspondence may be incorrect. Furthermore, if the cardinality of A and A' are not equal, there are actions without exact equivalences.

One option 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 depends on the transfer and reward functions in the two MDPs.[4] Figure 5 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 et al., 2007b), such that any novel actions in the target task are ignored. Consider adding an action in a mountain car target task, `pull hand brake`, which did not have an analog in the source task. The partial mapping could map `Forward` to `Forward`, and `Backward` to `Backward`, but not map `pull hand brake` to any source task action. 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).

It is possible that mappings between states, rather than between state variables, could be used for transfer, although no work has currently explored this formulation.[5] Another possible extension is to link the mappings rather than making them independent. For instance, the action mapping could depend on the state that the agent is in, or the state variable mapping could depend on the action

4. 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 the mappings may be one-to-many, one-to-one, or many-to-many.
5. However, there are many possibilities for using this approach for transfer learning, such as through bisimulation (see Section 9).

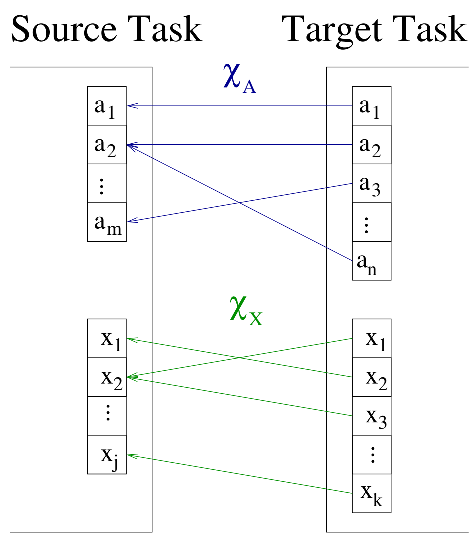

Figure 5: χ_A and χ_X are independent mappings that describe similarities between two MDPs. These mappings describe how actions in the target task are similar to actions in the source task and how state variables in the target task are similar to state variables in the source task, respectively.

selected. Though these extensions may be necessary based on the demands of particular MDPs, current methods have functioned well in a variety of tasks without such enhancements.

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. Work that attempts to learn a mapping that can be effectively used for transfer is discussed in Section 8.

3.5 Related Paradigms

In this survey, we consider transfer learning algorithms that use one or more source tasks to better learn in a different, but related, target task. There is a wide range of methods designed to improve the learning speed of RL methods. This section discusses four alternate classes of techniques for speeding up learning and differentiates them from transfer. While some TL algorithms may reasonably fit into one or more of the following categories, we believe that enumerating the types of methods *not* surveyed in this article will help clarify our subject of interest.

3.5.1 Lifelong Learning

Thrun (1996) suggested the notion of lifelong learning where an agent may experience a sequence of tasks. Others (cf., Sutton et al., 2007) later extended this idea to the RL setting, suggesting than an agent interacting with the world for an extended period of time will necessarily have to perform in a sequence of tasks. Alternately, the agent may discover a series of spatially, rather than temporally, separated sub-tasks. Transfer would be a key component of any such system, but the lifelong learning framework is more demanding than that of transfer. First, transfer algorithms may reasonably focus on transfer between a single pair of related tasks, rather than attempting to account for any future task that an agent could encounter. Second, 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 MDP (i.e., the real world).

3.5.2 IMITATION LEARNING

The primary motivations for imitation methods are to allow agents to learn by watching another agent with similar abilities (Price and Boutilier, 2003; Syed and Schapier, 2007) or a human (Abbeel and Ng, 2005; Kolter et al., 2008) perform a task. Such algorithms attempt to learn a policy by observing an outside actor, potentially improving upon the inferred policy. In contrast, our definition of transfer learning focuses on agents successfully reusing internal knowledge on novel problems.

3.5.3 HUMAN ADVICE

There is a growing body of work integrating human advice into RL learners. For instance, a human may provide action suggestions to the agent (cf., Maclin and Shavlik, 1996; Maclin et al., 2005) or guide the agent through on-line feedback (cf., Knox and Stone, 2008). Leveraging humans' background and task-specific knowledge can significantly improve agents' learning ability, but it relies on a human being tightly integrated into the learning loop, providing feedback in an on-line manner. This survey instead concentrates on transfer methods in which a human is not continuously available and agents must learn autonomously.

3.5.4 SHAPING

Reward shaping (Colombetti and Dorigo, 1993; Mataric, 1994) in an RL context typically refers to allowing agent to train on an artificial reward signal rather than R. For instance, in the mountain car task, the agent could be given a higher reward as it gets closer to the goal state, rather then receiving -1 at every state except the goal. However, if the human can compute such a reward, s/he would probably already know the goal location, knowledge that the agent typically does not have. Additionally, the constructed reward function must be a potential function. If it is not, the optimal policy for the new MDP could be different from that of the original (Ng et al., 1999). A second definition of shaping follows Skinner's research (Skinner, 1953) where the reward function is modified over time in order to direct the behavior of the learner. This method, as well as the approach of using a static artificial reward, are ways of injecting human knowledge into the task definition to improve learning efficacy.

Erez and Smart (2008) have argued for a third definition of shaping as any supervised, iterative, process to assist learning. This includes modifying the dynamics of the task over time, modifying the internal learning parameters over time, increasing the actions available to the agent, and extending the agent's policy time horizon (e.g., as done in value iteration). All of these methods rely on a human to intelligently assist the agent in its learning task and may leverage transfer-like methods to successfully reuse knowledge between slightly different tasks. When discussing transfer, we will emphasize how knowledge is successfully reused rather than how a human may modify tasks to achieve the desired agent behavior improve agent learning performance.

3.5.5 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. Taylor and Stone (2007a) examine the

Citation	Allowed Task Differences	Source Task Selection	Task Mappings	Transferred Knowledge	Allowed Learners	TL Metrics
Same state variables and actions: Section 4						
Selfridge et al. (1985)	t	h	N/A	Q	TD	tt†
Asada et al. (1994)	s_i	h	N/A	Q	TD	tt
Singh (1992)	r	all	N/A	Q	TD	ap, tr
Atkeson and Santamaria (1997)	r	all	N/A	model	MB	ap, j, tr
Asadi and Huber (2007)	r	h	N/A	π_p	H	tt
Andre and Russell (2002)	r, s	h	N/A	π_p	H	tr
Ravindran and Barto (2003b)	s, t	h	N/A	π_p	TD	tr
Ferguson and Mahadevan (2006)	r, s	h	N/A	pvf	Batch	tt
Sherstov and Stone (2005)	s_f, t	mod	N/A	A	TD	tr
Madden and Howley (2004)	s, t	all	N/A	rule	TD	tt, tr
Lazaric (2008)	s, t	lib	N/A	I	Batch	j, tr

Table 3: This table reproduces the first group of methods from Table 1.

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; past experience may be transferred rather than discarded if a new representation is desired. A more important benefit is that changing representations partway through learning can allow agents to achieve better performance in less time. Selecting a representation is often key for solving a problem (cf., 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. However, representation selection is a difficult problem in RL in general and discussions of representation selection (or its applications to transfer efficacy) are beyond the scope of this article.

4. Transfer Methods for Fixed State Variables and Actions

To begin our survey of TL methods, we examine the first group of methods in Table 1, reproduced in Table 3. These techniques may be used for transfer when the source and target tasks use the same state variables and when agents in both tasks have the same set of actions (see Figure 6).

In one of the earliest TL works for RL, Selfridge et al. (1985) demonstrated that it was faster to learn to balance a pole on a cart by changing the task's transition function, T, over time. The learner was first trained on a long and light pole. Once it successfully learned to balance the pole the task was made harder: the pole was shortened and made heavier. The total time spent training on a sequence of tasks and reusing the learned function approximator was faster than training on the hardest task directly.[6]

Similarly, the idea of *learning from easy missions* (Asada et al., 1994) also relies on a human constructing a set of tasks for the learner. In this work, the task (for example, a maze) is made incrementally harder not by changing the dynamics of the task, but by moving the agent's initial

6. As discussed in Section 3.5.3, we classify this work as transfer rather than as a "human advice" method; while the human may assist the agent in task selection, s/he does not provide direct on-line feedback while the agent learns.

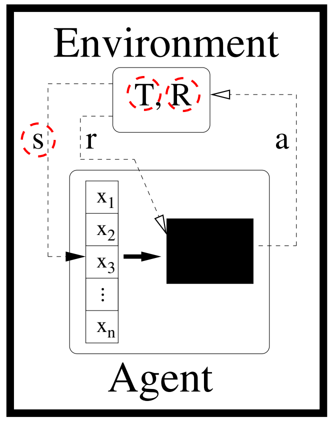

Figure 6: Methods in Section 4 are able to transfer between tasks that have different state spaces, different transition functions, and different reward functions, but only if the source and target tasks have the same actions and state variables. Dashed circles indicate the MDP components which may differ between the source task and target task.

state, $s_{initial}$, further and further from the goal state. The agent incrementally learns how to navigate to the exit faster than if it had tried to learn how to navigate the full maze directly. This method relies on having a known goal state from which a human can construct a series of source tasks of increasing difficulty.

Selfridge et al. (1985) and Asada et al. (1994) provide useful methods for improving learning, which follow from Skinner's animal training work. While they require a human to be in the loop, and to understand the task well enough to provide the appropriate guidance to the learner, these methods are relatively easy ways to leverage human knowledge. Additionally, they may be combined with many of the transfer methods that follow.

Rather than change a task over time, one could consider breaking down a task into a series of smaller tasks. This approach can be considered a type of transfer in that a single large target task can be treated as a series of simpler source tasks. Singh (1992) uses a technique he labels *compositional learning* to discover how to separate temporally sequential subtasks in a monolithic task. Each subtask has distinct beginning and termination conditions, and each subtask will be significantly easier to learn in isolation than in the context of the full task. Only the reward function, R, is allowed to change between the different subtasks and none of the other MDP components may vary, but the total reward can be increased. If subtasks in a problem are recognizable by state features, such subtasks may be automatically identified via vision algorithms (Drummond, 2002). Again, breaking a task into smaller subtasks can improve both the total reward and the asymptotic performance. This particular method is only directly applicable to tasks in which features clearly define subtasks due to limitations in the vision algorithm used. For instance, in a 2D navigation task each room may be a subtask and the steep value function gradient between impassable walls is easily identifiable. However, if the value function gradient is not distinct between different subtasks, or the subtask regions of state space are not polygonal, the algorithm will likely fail to automatically identify subtasks.

In Atkeson and Santamaria (1997), transfer between tasks in which only the reward function can differ are again considered. Their method successfully transfers a locally weighted regression model of the transition function, which is learned in a source task, by directly applying it to a target task. Because their model enables planning over the transition function and does not account for the reward function, they show significant improvement to the jumpstart and total reward, as well as the asymptotic performance.

The next three methods transfer partial policies, or options, between different tasks. First, Asadi and Huber (2007) have the agent identify states that "locally form a significantly stronger 'attractor' for state space trajectories" as subgoals in the source task (i.e., a doorway between rooms that is visited relatively often compared to other parts of the state space). The agent then learns options to reach these subgoals via a learned action-value function, termed the *decision-level* model. A second action-value function, the *evaluation-level* model, includes all actions and the full state space. The agent selects actions by only considering the decision-level model but uses discrepancies between the two models to automatically increase the complexity of the decision-level model as needed. The model is represented as a *Hierarchical Bounded Parameter SMDP*, constructed so that the performance of an optimal policy in the simplified model will be within some fixed bound of the performance of the optimal policy on the initial model. Experiments show that transferring both the learned options and the decision-level representation allow the target task agent to learn faster on a task with a different reward function. In the roughly 20,000 target task states, only 81 distinct states are needed in the decision-level model, as most states do not need to be distinguished when selecting from learned options.

Second, Andre and Russell (2002) transfer learned subroutines between tasks, which are similar to options. The authors assume that the source and target tasks have a hierarchical structure, such as in the *taxi domain* (Dietterich, 2000). On-line analysis can uncover similarities between two tasks if there are only small differences in the state space (e.g., the state variables do not change) and then directly copy over the subroutine, which functions as a partial policy, thereby increasing the total reward in the target task. This method highlights the connection between state abstraction and transfer; if similarities can be found between parts of the state space in the two tasks, it is likely that good local controllers or local policies can be directly transferred.

Third, Ravindran and Barto (2003b) learn *relativized options* in a small, human selected source task. When learning in the target task, the agent is provided these options and a set of possible transformations it could apply to them so that they were relevant in the target task. For instance, if the source task were a small grid navigation task, the target task could be a large grid composed of rooms with similar shape to the source task and the transformations could be rotation and reflection operators. The agent uses experience in the target and Bayesian parameter estimation to select which transformations to use so that the target task's total reward is increased. Learning time in the source task is ignored, but is assumed to be small compared to the target task learning time.

Next, Ferguson and Mahadevan (2006) take a unique approach to transfer information about the source task's structure. *Proto-value functions* (PVFs) (Mahadevan and Maggioni, 2007) specify an ortho-normal set of basis functions, without regard to R, which can be used to learn an action-value function. After PVFs are learned in a small source task, they can be transferred to another discrete MDP that has a different goal or small changes to the state space. The target task can be learned faster and achieve higher total reward with the transferred PVFs than without. Additionally, the PVF can be scaled to larger tasks. For example, the target maze could have twice the width and height of the source maze: R, S, and T are all scaled by the same factor. In all cases only the target task time is counted.

The goal of learning PVFs is potentially very useful for RL in general and TL in particular. It makes intuitive sense that high-level information about how to best learn in a domain, such as appropriate features to reason over, may transfer well across tasks. There are few examples of meta-learners where TL algorithms learn high level knowledge to assist the agent in learning, rather than lower-level knowledge about how to act. However, we believe that there is ample room for such methods, including methods to learn other domain-specific learning parameters, such as learning rates, function approximator representations, an so on.

Instead of biasing the target task agent's learning representation by transferring a set of basis functions, Sherstov and Stone (2005) consider how to bias an agent by transferring an appropriate action set. If tasks have large action sets, all actions could be considered when learning each task, but learning would be much faster if only a subset of the actions needed to be evaluated. If a reduced action set is selected such that using it could produce near-optimal behavior, learning would be much faster with very little loss in final performance. The standard MDP formalism is modified so that the agent reasons about *outcomes* and *classes*. Informally, rather than reasoning over the probability of reaching a given state after an action, the learner reasons over the actions' effect, or outcome. States are grouped together in classes such that the probability of a given outcome from a given action will be the same for any state in a class. The authors then use their formalism to bound the value lost by using their abstraction of the MDP. If the source and target are very similar, the source task can be learned with the full action set, the optimal action set can be found from the learned Q-values, and learning the target with this smaller action set can speed up learning in the target task. The authors also introduce *random task perturbation* (RTP) which creates a *series* of source tasks from a single source task, thereby producing an action set which will perform well in target tasks that are less similar to the source task. Transfer with and without RTP is experimentally compared to learning without transfer. While direct action transfer can perform worse than learning without transfer, RTP was able to handle misleading source task experience so that performance was improved relative to no transfer in all target tasks and performance using the transferred actions approaches that of the optimal target task action set. Performance was judged by the total reward accumulated in the target task. Leffler et al. (2007) extends the work of Sherstov and Stone by applying the outcome/class framework to learn a *single* task significantly faster, and provides empirical evidence of correctness in both simulated and physical domains.

The idea of RTP is not only unique in this survey, but it is also potentially a very useful idea for transfer in general. While a number of TL methods are able to learn from a set of source tasks, no others attempt to automatically generate these source tasks. If the goal of an agent is perform as well as possible in a novel target task, it makes sense that the agent would try to train on many source tasks, even if they are artificial. How to best generate such source tasks so that they are most likely to be useful for an arbitrary target task in the same domain is an important area of open research.

Similar to previously discussed work (Selfridge et al., 1985; Asada et al., 1994), *Progressive RL* (Madden and Howley, 2004) is a method for transferring between a progression of tasks of increasing difficulty, but is limited to discrete MDPs. After learning a source task, the agent performs *introspection* where a symbolic learner extracts rules for acting based on learned Q-values from all previously learned tasks. The RL algorithm and introspection use different state features. Thus the two learning mechanisms learn in different state spaces, where the state features for the symbolic learner are higher-level and contain information otherwise hidden from the agent. When the agent acts in a novel task, the first time it reaches a novel state it initialize the Q-values of that state so that the action suggested by the learned rule is preferred. Progressive RL allows agents to learn information in a set of tasks and then abstract the knowledge to a higher-level representation, allowing the agent to achieve higher total reward and reach the goal state for the first time faster. Time spent in the source task(s) is not counted.

Finally, Lazaric (2008) demonstrates that source task *instances* can be usefully transferred between tasks. After learning one or more source tasks, some experience is gathered in the target task, which may have a different state space or transition function. Saved instances (that is, observed $\langle s,a,r,s'\rangle$ tuples) are compared to instances from the target task. Instances from the source tasks that are most similar, as judged by their distance and alignment with target task data, are transferred. A batch learning algorithm then uses both source instances and target instances to achieve a higher reward and a jumpstart. *Region transfer* takes the idea one step further by looking at similarity with the target task per-sample, rather than per task. Thus, if source tasks have different regions of the state space which are more similar to the target, only those most similar regions can be transferred. In these experiments, time spent training in the target task is not counted towards the TL algorithm.

Region transfer is the only method surveyed which explicitly reasons about task similarity in *different parts* of the state space, and then selects source task(s) to transfer from. In domains where target tasks have regions of the state space that are similar to one or more source tasks, and other areas which are similar to other source tasks (or are similar to no source tasks), region transfer may provide significant performance improvements. As such, this method provides a unique approach to measuring, and exploiting, task similarity on-line. It is likely that this approach will inform future transfer methods, and is one possible way of accomplishing step # 1 in Section 2: Given a target task, select an appropriate source task from which to transfer, if one exists.

Taken together, these TL methods show that it is possible to efficiently transfer many different types of information between tasks with a variety of differences. It is worth re-emphasizing that many TL methods may be combined with other speedup methods, such as reward shaping, or with other transfer methods. For instance, when transferring between maze tasks, basis functions could be learned (Ferguson and Mahadevan, 2006) in the source task, a set of actions to transfer could be selected after training on a set of additional generated source tasks (Sherstov and Stone, 2005), and then parts of different source tasks could be leveraged to learn a target task (Lazaric, 2008). A second example would be to start with a simple source task and change it over time by modifying the transition function (Selfridge et al., 1985) and start state (Asada et al., 1994), while learning options (Ravindran and Barto, 2003b), until a difficult target task is learned. By examining how the source and target task differ and what base learning method is used, RL practitioners may select one or more TL method to apply to their domain of interest. However, in the absence of theoretical guarantees of transfer efficacy, any TL method has the potential to be harmful, as discussed further in Section 9.2.

5. Multi-Task Learning Methods

This section discusses scenarios where the source tasks and target task have the same state variables and actions. However, these methods (see Table 4, reproduced from Table 1) are explicitly MTL, and all methods in this section are designed to use multiple source tasks (see Figure 7). Some methods leverage all experienced source tasks when learning a novel target task and others are able to choose a subset of previously experienced tasks. Which approach is most appropriate depends on the assumptions about the task distribution: if tasks are expected to be similar enough that all past experience is useful, there is no need to select a subset. On the other hand, if the distribution of

Citation	Allowed Task Differences	Source Task Selection	Task Mappings	Transferred Knowledge	Allowed Learners	TL Metrics
Multi-Task learning: Section 5						
Mehta et al. (2008)	r	lib	N/A	π_p	H	tr
Perkins and Precup (1999)	t	all	N/A	π_p	TD	tt
Foster and Dayan (2004)	s_f	all	N/A	sub	TD, H	j, tr
Fernandez and Veloso (2006)	s_i, s_f	lib	N/A	π	TD	tr
Tanaka and Yamamura (2003)	t	all	N/A	Q	TD	j, tr
Sunmola and Wyatt (2006)	t	all	N/A	pri	B	j, tr
Wilson et al. (2007)	r, s_f	all	N/A	pri	B	j, tr
Walsh et al. (2006)	r, s	all	N/A	fea	any	tt
Lazaric (2008)	r	all	N/A	fea	Batch	ap, tr

Table 4: This table reproduces the group of MTL methods from Table 1.

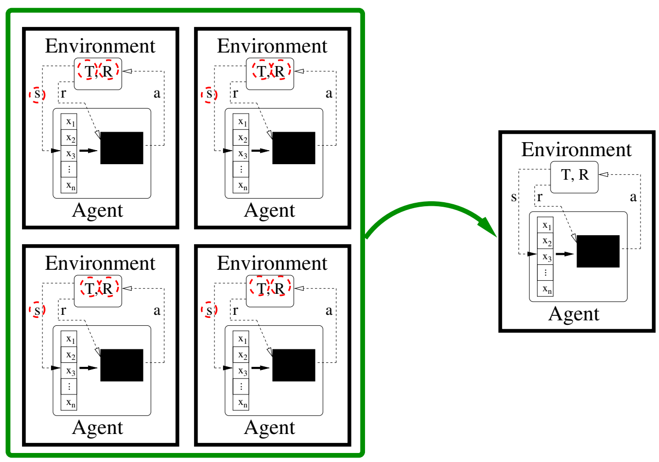

Figure 7: Multi-task learning methods assume tasks are chosen from a fixed distribution, use one or more source tasks to help learn the current task, and assume that all the tasks have the same actions and state variables. Dashed circles indicate the MDP components which may differ between tasks.

tasks is multi-modal, it is likely that transferring from all tasks is sub-optimal. None of the methods account for time spent learning in the source task(s) as the primary concern is effective learning on the next task chosen at random from an unknown (but fixed) distribution of MDPs.

Variable-reward hierarchical reinforcement learning (Mehta et al., 2008) assumes that the learner will train on a sequence of tasks which are identical except for different *reward weights*. The reward weights define how much reward is assigned via a linear combination of *reward features*. The authors provide the reward features to the agent for a given set of tasks. For instance, in a real-time strategy domain different tasks could change the reward features, such as the benefit from collecting units of gold or from damaging the enemy. However, it is unclear how many domains of interest have reward features, which are provided to the agent at the start of each task. Using a hierarchical RL method, subtask policies are learned. When a novel target task is encountered, the agent sets the initial policy to that of the most similar source task, as determined by the dot product with previously observed reward weight vectors. The agent then uses an ϵ-greedy action selection method at each level of the task hierarchy to decide whether to use the best known sub-task policy or explore. Some sub-tasks, such as navigation, will never need to be relearned for different tasks because they are unaffected by the reward weights, but any suboptimal sub-task policies will be improved. As the agent experiences more tasks, the total reward in each new target task increases, relative to learning the task without transfer.

A different problem formulation is posed by Perkins and Precup (1999) where the transition function, T, may change after reaching the goal. Upon reaching the goal, the agent is returned to the start state and is not told if, or how, the transition function has changed, but it knows that T is drawn randomly from some fixed distribution. The agent is provided a set of hand-coded options which assist in learning on this set of tasks. Over time, the agent learns an accurate action-value function over these options. Thus, a single action-value function is learned over a set of tasks, allowing the agent to more quickly reach the goal on tasks with novel transition functions.

Instead of transferring options, Foster and Dayan (2004) aim to identify sub-tasks in a source task and use this information in a target task, a motivation similar to that of Singh (1992). Tasks are allowed to differ in the placement of the goal state. As optimal value functions are learned in source tasks, an *expectation-maximization* algorithm (Dempster et al., 1977) identifies different "fragmentations," or sub-tasks, across all learned tasks. Once learned, the fragmentations are used to augment the state of the agent. Each sub-problem can be learned independently; when encountering a new task, much of the learning is already complete because the majority of sub-problems are unchanged. The fragmentations work with both a flat learner (i.e., TD) and an explicitly hierarchical learner to improve the jumpstart and total reward.

Probabilistic policy reuse (Fernandez and Veloso, 2006) also considers a distribution of tasks in which only the goal state differs, but is one of the most robust MTL methods in terms of appropriate source task selection. Although the method allows a single goal state to differ between the tasks, it requires that S, A, and T remain constant. If a newly learned policy is significantly different from existing policies, it is added to a policy library. When the agent is placed in a novel task, on every timestep, it can choose to: exploit a learned source task policy, exploit the current best policy for the target task, or randomly explore. If the agent has multiple learned policies in its library, it probabilistically selects between policies so that over time more useful policies will be selected more often. While this method allows for probabilistic mixing of the policies, it may be possible to treat the past policies as options which can be executed until some termination condition is met, similar to a number of previously discussed methods. By comparing the relative benefits of mixing past policies and treating them as options, it may be possible to better understand when each of the two approaches is most useful.

The idea of constructing an explicit policy library is likely to be useful in future TL research, particularly for agents that train on a number of source tasks that have large qualitative differences (and thus very different learned behaviors). Although other methods also separately record information from multiple source tasks (cf., Mehta et al., 2008; Lazaric, 2008), Fernandez and Veloso explicitly reason about the library. In addition to reasoning over the amount of information stored, as a function of number and type of source tasks, it will be useful to understand how many target task samples are needed to select the most useful source task(s).

Unlike probabilistic policy reuse, which selectively transfers information from a single source task, Tanaka and Yamamura (2003) gather statistics about *all* previous tasks and use this amalgamated knowledge to learn novel tasks faster. Specifically, the learner keeps track of the average and the deviation of the action value for each (s,a) pair observed in all tasks. When the agent encounters a new task, it initializes the action-value function so that every (s,a) pair is set to the current average for that pair, which provides a benefit relative to uninformed initialization. As the agent learns the target task with Q-learning and prioritized sweeping,[7] the agent uses the standard deviation of states' Q-values to set priorities on TD backups. If the current Q-value is far from the average for that (s,a) pair, its value should be adjusted more quickly, since it is likely incorrect (and thus should be corrected before affecting other Q-values). Additionally, another term accounting for the variance within individual trials is added to the priority; Q-values that fluctuate often within a particular trial are likely wrong. Experiments show that this method, when applied to sets of discrete tasks with different transition functions, can provide significant improvement to jumpstart and total reward.

The next two methods consider how priors can be effectively learned by a Bayesian MTL agent. First, Sunmola and Wyatt (2006) introduce two methods that use instances from source tasks to set priors in a Bayesian learner. Both methods constrain the probabilities of the target task's transition function by using previous instances as a type of prior. The first method uses the working prior to generate possible models which are then tested against data in the target task. The second method uses a probability perturbation method in conjunction with observed data to improve models generated by the prior. Initial experiments show that the jumpstart and total reward can be improved if the agent has an accurate estimation of the prior distributions of the class from which the target is drawn. Second, Wilson et al. (2007) consider learning in a hierarchical Bayesian RL setting. Setting the prior for Bayesian models is often difficult, but in this work the prior may be transferred from previously learned tasks, significantly increasing the learning rate. Additionally, the algorithm can handle "classes" of MDPs, which have similar model parameters, and then recognize when a novel class of MDP is introduced. The novel class may then be added to the hierarchy and a distinct prior may be learned, rather than forcing the MDP to fit into an existing class. The location of the goal state and the parameterized reward function may differ between the tasks. Learning on subsequent tasks shows a clear performance improvement in total reward, and some improvement in jumpstart.

While Bayesian methods have been shown to be successful when transferring between classification tasks (Roy and Kaelbling, 2007), and in non-transfer RL (Dearden et al., 1999), only the two methods above use it in RL transfer. The learner's bias is important in all machine learning settings. However, Bayesian learning makes such bias explicit. Being able to set the bias through transfer from similar tasks may prove to be a very useful heuristic—we hope that additional transfer methods will be developed to initialize Bayesian learners from past tasks.

Walsh et al. (2006) observe that "deciding what knowledge to transfer between environments can be construed as determining the correct state abstraction scheme for a set of source [tasks] and then applying this compaction to a target [task]." Their suggested framework solves a set of MDPs, builds abstractions from the solutions, extracts relevant features, and then applies the feature-based abstraction function to a novel target task. A simple experiment using tasks with different state spaces and reward functions shows that the time to learn a target task is decreased by using MTL. Building upon their five defined types of state abstractions (as defined in Li et al. 2006), they give theoretical results showing that when the number of source tasks is large (relative to the differences

7. Prioritized sweeping (Moore and Atkeson, 1993) is an RL method that orders adjustments to the value function based on their "urgency," which can lead to faster convergence than when updating the value function in the order of visited states.

Citation	Allowed Task Differences	Source Task Selection	Task Mappings	Transferred Knowledge	Allowed Learners	TL Metrics
Different state variables and actions – no explicit task mappings: Section 6						
Konidaris and Barto (2006)	p	h	N/A	R	TD	j, tr
Konidaris and Barto (2007)	p	h	N/A	π_p	TD	j, tr
Banerjee and Stone (2007)	a, v	h	N/A	fea	TD	ap, j, tr
Guestrin et al. (2003)	#	h	N/A	Q	LP	j
Croonenborghs et al. (2007)	#	h	N/A	π_p	RRL	ap, j, tr
Ramon et al. (2007)	#	h	N/A	Q	RRL	ap, j, tt†, tr
Sharma et al. (2007)	#	h	N/A	Q	TD, CBR	j, tr

Table 5: This table reproduces the third group of methods from Table 1.

between the different tasks), four of the five types of abstractions are guaranteed to produce the optimal policy in a target task using Q-learning.

Similar to Walsh et al. (2006), Lazaric (2008) also discovers features to transfer. Rather than learning tasks sequentially, as in all the papers above, one could consider learning different tasks in parallel and using the shared information to learn the tasks better than if each were learned in isolation. Specifically, Lazaric (2008) learns a set of tasks with different reward functions using the batch method *Fitted Q-iteration* (Ernst et al., 2005). By leveraging a multi-task feature learning algorithm (Argyrious et al., 2007), the problem can be formulated as a joint optimization problem to find the best features and learning parameters across observed data in all tasks. Experiments demonstrate that this method can improve the total reward and can help the agent to ignore irrelevant features (i.e., features which do not provide useful information). Furthermore, since it may be possible to learn a superior representation, asymptotic performance may be improved as well, relative to learning tasks in isolation.

The work in this section, as summarized in the second section of Table 1, explicitly assumes that all MDPs an agent experiences are drawn from the same distribution. Different tasks in a single distribution could, in principal, have different state variables and actions, and future work should investigate when allowing such flexibility would be beneficial.

6. Transferring Task-Invariant Knowledge Between Tasks with Differing State Variables and Actions

This section, unlike the previous two, discusses methods that allow the source task and target task to have different state variables and actions (see Figure 8 and the methods in Table 5). These methods formulate the problem so that no explicit mapping between the tasks is needed. Instead the agent reasons over abstractions of the MDP that are invariant when the actions or state variables change.

For example, Konidaris and Barto (2006) have separated the standard RL problem 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), although such a space may be non-Markovian.[8] The problem-space, on the other hand, may change between source and target

8. 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.

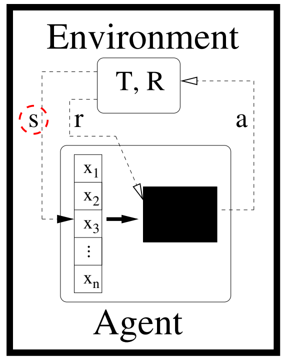

Figure 8: Methods in Section 6 are able to transfer between tasks with different state spaces. Although T, R, A, and the state variables may also technically change, the agent's internal representation is formulated so that they remain fixed between source and target tasks. MDP components with a dashed circle may change between the source task and target task.

problems and is assumed to be Markovian. The authors' method learns a shaping reward on-line in agent-space while learning a source task. If a later target task has a similar reward structure and action set, the learned shaping reward will help the agent achieve a jumpstart and higher total reward. For example, suppose that one of the agent's sensors measures the distance between it and a particular important state (such as a beacon located near the goal state). The agent may learn a shaping reward that assigns reward when the state variable describing its distance to the beacon is reduced, even in the absence of an environmental reward. The authors assume that there are no novel actions (i.e., actions which are not in the source task's problem-space) but any new state variables can be handled if they can be mapped from the novel problem-space into the familiar agent-space. Additionally, 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." Determining whether or not a sequence of tasks meet this criterion is left for future work.

In later work (Konidaris and Barto, 2007), the authors assume knowledge of "pre-specified salient events," which make learning options tractable. While it may be possible to learn options without requiring such events to be specified, the paper focuses on how to use such options rather than option learning. Specifically, when the agent achieves one of these subgoals, such as unlocking a door or moving through a doorway, it may learn an option to achieve the event again in the future. As expected, problem-space options speed up learning a single task. More interesting, when the agent trains on a series of tasks, options in both agent-space and problem-space significantly increase the jumpstart and total reward in the target task (time spent learning the source task is discounted). The authors suggest that agent-space options will likely be more portable than problem-space options in cases where the source and target tasks are less similar—indeed, problem-space options will only be portable when source and target tasks are very similar.

In our opinion, agent- and problem-space are ideas that should be further explored as they will likely yield additional benefits. 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.

If transfer is applied to game trees, changes in actions and state variables may be less problematic. Banerjee and Stone (2007) are able to transfer between games by focusing on this more abstract formulation. For instance, in experiments the learner identified the concept of a *fork*, a state where the player could win on the subsequent turn regardless of what move the opponent took next. After training in the source task, analyzing the source task data for such features, and then setting the value for a given feature based on the source task data, such features of the game tree were used in a variety of target tasks. This analysis focuses on the effects of actions on the game tree and thus the actions and state variables describing the source and target game can differ without requiring an inter-task mapping. Source task time is discounted, but jumpstart, total reward, and asymptotic performance are all improved via transfer. Although the experiments in the paper use only temporal difference learning, it is likely that this technique would work well with other types of learners.

Guestrin et al. (2003) examine a similar problem in the context of planning in what they term a *relational MDP*. Rather than learning a standard value function, an agent-centered value function for each *class* of agents is calculated in a source task, forcing all agents of a given class type to all have the same value function. However, these class value functions are defined so that they are independent of the number of agents in a task, allowing them to be directly used in a target task which has additional (or fewer) agents. No further learning is done in the target task, but the transferred value functions perform better than a handcoded strategy provided by the authors, despite having additional friendly and adversarial agents. However, the authors note that the technique will not perform well in heterogeneous environments or domains with "strong and constant interactions between many objects."

Relational Reinforcement Learning may also be used for effective transfer. Rather than reasoning about states as input from an agent's sensors, an RRL learner typically reasons about a state in propositional form by constructing first-order rules. The learner can easily abstract over specific object identities as well as the number of objects in the world; transfer between tasks with different number of objects is simplified. For instance, Croonenborghs et al. (2007) first learn a source task policy with RRL. The learned policy is used to create examples of state-action pairs, which are then used to build a relational decision tree. This tree predicts, for a given state, which action would be executed by the policy. Lastly, the trees are mined to produce *relational options*. These options are directly used in the target task with the assumption that the tasks are similar enough that no translation of the relational options is necessary. The authors consider three pairs of source/target tasks where relational options learned in the source directly apply to the target task (only the number of objects in the tasks may change), and learning is significantly improved in terms of jumpstart, total reward, and asymptotic performance.

Other work using RRL for transfer (Ramon et al., 2007) introduces the TGR algorithm, a relational decision tree algorithm. TGR incrementally builds a decision tree in which internal nodes use first-order logic to analyze the current state and where the tree's leaves contain action-values. The algorithm uses four tree-restructuring operators to effectively use available memory and increase sample efficacy. Both target task time and total time are reduced by first training on a simple source task and then on a related target task. Jumpstart, total reward, and asymptotic performance also appear to improve via transfer.

RRL is a particularly attractive formulation in the context of transfer learning. In RRL, agents can typically act in tasks with additional objects without reformulating their, although additional training may be needed to achieve optimal (or even acceptable) performance levels. When it is possible to frame a domain of interest as an RRL task, transfer between tasks with different numbers of objects or agents will likely be relatively straightforward.

With motivation similar to that of RRL, some learning problems can be framed so that agents choose between high-level actions that function regardless of the number of objects being reasoned about. Sharma et al. (2007) combines case-based reasoning with RL in the *CAse-Based Reinforcement Learner* (CARL), a multi-level architecture includes three modules: a planner, a controller, and a learner. The tactical layer uses the learner to choose between high-level actions which are independent of the number of objects in the task. The cases are indexed by: high-level state variables (again independent of the number of objects in the task), the actions available, the Q-values of the actions, and the cumulative contribution of that case on previous timesteps. Similarity between the current situation and past cases is determined by Euclidean distance. Because the state variables and actions are defined so that the number of objects in the task can change, the source and target tasks can have different numbers of objects (in the example domain, the authors use different numbers of player and opponent troops in the source and target tasks). Time spent learning the source task is not counted, but the target task performance is measured in terms of jumpstart, *asymptotic gain* (a metric related to the improvement in average reward over learning), and *overall gain* (a metric based on the total reward accrued).

In summary, methods surveyed in this section all allow transfer between tasks with different state variables and actions, as well as transfer functions, state spaces, and reward functions. By framing the task in an agent-centric space, limiting the domain to game trees, or using a learning method that reasons about variable numbers of objects, knowledge can be transferred between tasks with relative ease because problem representations do not change from the learner's perspective. In general, not all tasks may be formulated so that they conform to the assumptions made by TL methods presented in this section.

7. Explicit Mappings to Transfer between Different Actions and State Representations

This section of the survey focuses on a set of methods which are more flexible than those previously discussed as they allow the state variables and available actions to differ between source and target tasks (see Table 6 and Figure 9). All methods in this section use inter-task mappings, enabling transfer between pairs of tasks that could not be addressed by methods in the previous section. Note that because of changes in state variables and actions, R, S, and T, all technically change as well (they are functions defined over actions and state variables). However, as we elaborate below, some of the methods allow for significant changes in reward functions between the tasks, while most do not.

In Taylor et al. (2007a), the authors assume that a mapping between the source and target tasks is provided to the learner. The learner first trains in a source task using a value-function-learning method. Before learning begins in the target task, every action-value for each state in the target task is initialized via learned source task values. This work experimentally demonstrates that value-function transfer can cause significant speedup by transferring between tasks that have different state variables and actions. Additionally, different methods for performing the value-function transfer are examined, different function approximators are successfully used, and multi-step transfer is demonstrated (i.e., transfer from task A to task B to task C). This TL method demonstrates that when

Citation	Allowed Task Differences	Source Task Selection	Task Mappings	Transferred Knowledge	Allowed Learners	TL Metrics
Different state variables and actions – inter-task mappings used: Section 7						
Taylor et al. (2007a)	a, v	h	sup	Q	TD	tt†
Taylor et al. (2007b)	a, v	h	sup	π	PS	tt†
Taylor et al. (2008b)	a, v	h	sup	I	MB	ap, tr
Torrey et al. (2005) Torrey et al. (2006)	a, r, v	h	sup	rule	TD	j, tr
Torrey et al. (2007)	a, r, v	h	sup	π_p	TD	j, tr
Taylor and Stone (2007b)	a, r, v	h	sup	rule	any/TD	j, tt†, tr

Table 6: This table reproduces the fourth group of methods from Table 1.

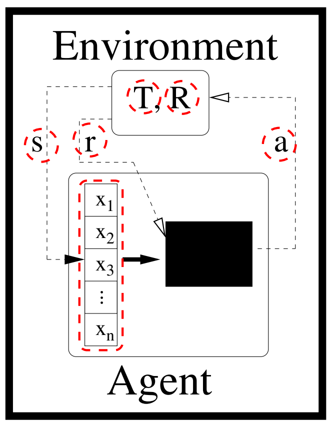

Figure 9: Methods in Section 7 focus on transferring between tasks with different state features, action sets, and possible reward functions (which, in turn, causes the state space and transition function to differ as well). As in previous figures, MDP components with a dashed circle may change between the source task and target task.

faced with a difficult task, it may be faster overall to first train on an artificial source task or tasks and then transfer the knowledge to the target task, rather than training on the target task directly. The authors provide no theoretical guarantees about their method's effectiveness, but hypothesize conditions under which their TL method will and will not perform well, and provide examples of when their method fails to reduce the training time via transfer.

In subsequent work, Taylor et al. (2007b) transfer entire policies between tasks with different state variables and actions, rather than action-value functions. A set of policies is first learned via a genetic algorithm in the source task and then transformed via inter-task mappings. Additionally, partial inter-task mappings are introduced, which may be easier for a human to intuit in many domains. Specifically, those actions and state variables in the target which have "very similar" actions and state variables in the source task are mapped, while novel state variables and actions in the target task are left unmapped. Policies are transformed using one of the inter-task mappings and then used to seed the learning algorithm in the target task. As in the previous work, this TL method can successfully reduce both the target task time and the total time.

Later, Taylor et al. (2008b) again consider pairs of tasks where the actions differ, the state variables differ, and inter-task mappings are available to the learner. In this work, the authors allow transfer between model-learning methods by transferring instances, which is similar in spirit to Lazaric (2008). Fitted R-MAX (Jong and Stone, 2007), an instance-based model-learning method capable of learning in continuous state spaces, is used as the base RL method, and source task instances are transferred into the target task to better approximate the target task's model. Experiments in a simple continuous domain show that transfer can improve the jumpstart, total reward, and asymptotic performance in the target task.

Another way to transfer is via learned *advice* or preferences. Torrey et al. (2005) automatically extract such advice from a source task by identifying actions which have higher Q-values than other available actions.[9] Such advice is mapped via human-provided inter-task mappings to the target task as preferences given to the target task learner. In this work, Q-values are learned via support vector regression, and then *Preference Knowledge Based Kernel Regression* (KBKR) (Maclin et al., 2005) adds the advice as soft constraints in the target, setting relative preferences for different actions in different states. The advice is successfully leveraged by the target task learner and decreases the target task learning time, even when the source task has different state variables and actions. Additionally, the reward structure of the tasks may differ substantially: their experiments use a source task whose reward is an unbounded score based on episode length, while the target task's reward is binary, depending on if the agents reached a goal state or not. Source task time is discounted and the target task learning is improved slightly in terms of total reward and asymptotic performance.

Later work (Torrey et al., 2006) improves upon this method by using *inductive logic programming* (ILP) to identify *skills* that are useful to the agent in a source task. A trace of the agent in the source task is examined and both positive and negative examples are extracted. Positive and negative examples are identified by observing which action was executed, the resulting outcome, the Q-value of the action, and the relative Q-value of other available actions. Skills are extracted using the ILP engine Aleph (Srinivasan, 2001) by using the F_1 score (the harmonic mean of precision and recall). These skills are then mapped by a human into the target task, where they improve learning via KBKR. Source task time is not counted towards the target task time, jumpstart may be improved, and the total reward is improved. The source and target tasks again differ in terms of state variables, actions, and reward structure. The authors also show how human-provided advice may be easily incorporated in addition to advice generated in the source task. Finally, the authors experimentally demonstrate that giving bad advice to the learner is only temporarily harmful and that the learner can "unlearn" bad advice over time, which may be important for minimizing the impact of negative transfer.

Torrey et al. (2007) further generalize their technique to transfer *strategies*, which may require composing several skills together, and are defined as a finite-state machine (FSM). The *structure learning* phase of their algorithm analyzes source task data to find sequences of actions that distinguish between successful and unsuccessful games (e.g., whether or not a goal was reached), and composes the actions into a FSM. The second phase, *ruleset learning*, learns when each action in the strategy should be taken based on state features, and when the FSM should transition to the next state. Experience in the source task is again divided into positive and negative sequences for Aleph. 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

9. While this survey focuses on automatically learned knowledge in a source task, rather than human-provided knowledge, Torrey et al. (2005) show that both kinds of knowledge can be effectively leveraged.

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 learner chooses from the MDP's actions, not the high-level strategies, and can learn to improve on the transferred strategies. Experiments demonstrate that strategy transfer significantly improves the jumpstart and total reward in the target task when the source and target tasks have different state variables and actions (source task time is again discounted).

Similar to strategy transfer, Taylor and Stone (2007b) learn *rules* with RIPPER (Cohen, 1995) that summarize a learned source task policy. The rules are then transformed via handcoded inter-task mappings so that they could 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," resulting in an improved jumpstart and total reward. By using rules as an intermediary between the two tasks, the authors argue that the source and target tasks can use different learning methods, effectively de-coupling the two learners. Similarities with Torrey et al. (2007) include a significant improvement in initial performance and no provision to automatically handle scale differences.[10] The methods differ primarily in how advice is incorporated into the target learner and the choice of rule learner.

Additionally, Taylor and Stone (2007b) demonstrated that *inter-domain* transfer is possible. The two source tasks in this paper were discrete, fully observable, and one was deterministic. The target task, however, had a continuous state space, was partially observable, and had stochastic actions. Because the source tasks required orders of magnitude less time, the total time was roughly equal to the target task time. Our past work has used the term "inter-domain transfer" for transfer between qualitatively different domains, such as between a board game and a soccer simulation. However, this term is not well defined, or even agreed upon in the community. For instance, Swarup and Ray (2006) use the term "cross-domain transfer" to describe the reuse of a neural network structure between classification tasks with different numbers of boolean inputs and a single output. However, our hope is that researchers will continue improve transfer methods so that they may usefully transfer from very dissimilar tasks, similar to the way that humans may transfer high level ideas between very different domains.

This survey has discussed examples of of low- and high-level knowledge transfer. For instance, learning general rules or advice may be seen as relatively high level, whereas transferring specific Q-values or observed instances is quite task-specific. Our intuition is that higher-level knowledge may be more useful when transferring between very dissimilar tasks. For instance, it is unlikely that Q-values learned for a checkers game will transfer to chess, but the concept of a fork may transfer well. This has not been definitely shown, however, nor is there a quantitative way to classify knowledge in terms of low- or high-level. We hope that future work will confirm or disconfirm this hypothesis, as well as generate guidelines as to when different types of transferred knowledge is most appropriate.

All methods in this section use some type of inter-task mapping to allow transfer between MDPs with very different specifications. While these results show that transfer can provide a significant benefit, they presuppose that the mappings are provided to the learner. The following section considers methods that work to autonomously learn such inter-task mappings.

10. To our knowledge, there is currently no published method to automatically scale rule constants. Such scaling would be necessary if, for instance, source task distances were measured in feet, but target task distances were measured in meters.

Citation	Allowed Task Differences	Source Task Selection	Task Mappings	Transferred Knowledge	Allowed Learners	TL Metrics
Learning inter-task mappings: Section 8						
Kuhlmann and Stone (2007)	a, v	h	T	Q	TD	j, tr
Liu and Stone (2006)	a, v	h	T	N/A	all	N/A
Soni and Singh (2006)	a, v	h	M_a, sv_g, exp	N/A	all	ap, j, tr
Talvitie and Singh (2007)	a, v	h	M_a, sv_g, exp	N/A	all	j
Taylor et al. (2007b)*	a, v	h	sv_g, exp	N/A	all	tt[†]
Taylor et al. (2008c)	a, v	h	exp	N/A	all	j, tr

Table 7: This table reproduces the group of inter-task learning methods from Table 1.

8. Learning 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 a mapping between tasks can be learned, such that source task knowledge may be exploited in a novel target task with different state variables and actions (see Figure 10 and the final group in Table 1). Note that in this section, all but one of the methods have N/A for transfer method—with the exception of Kuhlmann and Stone (2007), the papers covered in this section introduce mapping-learning methods and then use existing methods to validate the mapping efficacy.

One current challenge of TL research is to reduce the amount of information provided to the learner about the relationship between the source and target tasks. If a human is directing the learner through a series of tasks, the similarities (or analogies) between the tasks will likely be provided by the human's intuition. If transfer is to succeed in an autonomous setting, however, the learner must first determine how (and whether) two tasks are related, and only then may the agent leverage its past knowledge to learn in a target task. Learning task relationships is critical if agents are to transfer without human input, either because the human is outside the loop, or because the human is *unable* to provide similarities between tasks. Methods in this section differ primarily in what information must be provided. At one end of the spectrum, Kuhlmann and Stone (2007) assume that a complete description of R, S, and T are given, while at the other, Taylor et al. (2008c) learn the mapping exclusively from experience gathered via environmental interactions.

Given a complete description of a game (i.e., the full model of the MDP), Kuhlmann and Stone (2007) analyze the game to produce a *rule graph*, an abstract representation of a deterministic, full information game. A learner first trains on a series of source task games, storing the rule graphs and learned value functions. When a novel target task is presented to the learner, it first constructs the target task's rule graph and then attempts to find a source task that has an isomorphic rule graph. The learner assumes that a transition function is provided and uses value-function-based learning to estimate values for *afterstates* of games. Only state variables need to be mapped between source and target tasks, and this is exactly the mapping found by graph matching. For each state in the target task, initial Q-values are set by finding the value of the corresponding state in the source task. Three types of transfer are considered: direct, which copies afterstate values over without modification; inverse, which accounts for a reversed goal or switched roles; and average, with copies the average

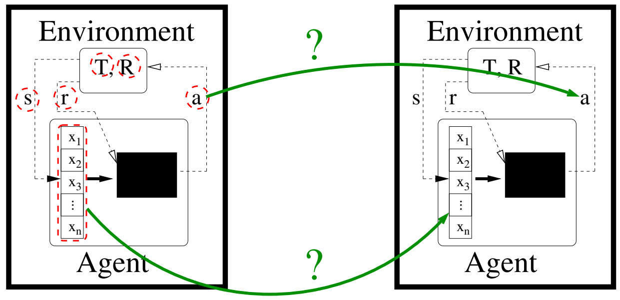

Figure 10: Section 8 presents methods to learn the relationship between tasks with different state variables and actions. As in previous figures, MDP components with a dashed circle may change between the source task and target task.

of a set of Q-values and can be used for boards with different sizes. Source task time is ignored but jumpstart and total reward can both be improved in the target task.

The previous work assumes full knowledge of a transition function. A more general approach could assume that the agent has only a qualitative understanding of the transition function. For instance, *qualitative dynamic Bayes networks* (QDBNs) (Liu and Stone, 2006), summarize the effects of actions on state variables but are not precise (for instance, they could not be used as a generative model for planning). If QDBNs are provided to an agent, a graph mapping technique can automatically find a mapping between actions and state variables in two tasks with relatively little computational cost. The authors show that mappings can be learned autonomously, effectively enabling value function transfer between tasks with different state variables and actions. However, it remains an open question as to whether or not QDBNs are learnable from experience, rather than being hand-coded.

The next three methods assume knowledge about how state variables are used to describe objects in a multi-player task. 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. First, Soni and Singh (2006) supply an agent with a series of possible state transformations and an inter-task action mapping. There is one such transformation, X, for every possible mapping of target task variables to source task variables. 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 $\pi_{source}(X(s))$. This method has a similar motivation to that of Fernandez and Veloso (2006), but here the authors are learning to select between possible mappings rather than possible previous policies. Over time the agent uses Q-learning to select the best state variable mapping as well as learn the action-values for the target task. The jumpstart, total reward, and asymptotic performance are all slightly improved when using this method, but its efficacy will be heavily dependent on the number of possible mappings between any source and target task.

Second, *AtEase* (Talvitie and Singh, 2007) also generates a number of possible state variable mappings. The action mapping is again assumed and the target task learner treats each of the possible mappings as an arm on a multi-armed bandit (Bellman, 1956). The authors prove their algorithm learns in time proportional to the number of possible mappings rather than the size of the problem: "in time polynomial in T, [the algorithm] accomplishes an actual return close to the asymptotic return of the best expert that has mixing time at most T." This approach focuses efficient selection of a proposed state variable mappings and does not allow target task learning.

Third, these assumptions are relaxed slightly by Taylor et al. (2007b), who show that it is possible to learn both the action and state variable mapping simultaneously by leveraging a classification technique, although it again relies on the pre-specified state variable groupings (i.e., knowing that "distance to teammate" refers to a teammate, but not which teammate). Action and state variable classifiers are trained using recorded source task data. For instance, the source task agent records s_{source}, a_{source}, s'_{source} tuples as it interacts with the environment. An action classifier is trained so that $C(s_{source,object}, s'_{source,object}) = a_{source}$ for each object present in the source task. Later, the target task agent again records s_{target}, a_{target}, s'_{target} tuples. Then the action classifier can again be used for to classify tuples for every target task object: $C(s_{target,object}, s'_{target,object}) = a_{source}$, where such a classification would indicate a mapping between a_{target} and a_{source}. Relatively little data is needed for accurate classification; the number of samples needed to learn in the target task far outweighs the number of samples used by the mapping-leaning step. While the resulting mappings are not always optimal for transfer, they do serve to effectively reduce target task training time as well as the total training time.

The MASTER algorithm (Taylor et al., 2008c) was designed to further relax the knowledge requirements of Taylor et al. (2007b): no state variable groupings are required. The key idea of MASTER is to save experienced source task instances, build an approximate transition model from a small set of experienced target task instances, and then test possible mappings offline by measuring the prediction error of the target-task models on source task data. This approach is sample efficient at the expense of high computational complexity, particularly as the number of state variables and actions increase. The method uses an exhaustive search to find the inter-task mappings that minimize the prediction error, but more sophisticated (e.g., heuristic) search methods could be incorporated. Experiments show that the learned inter-task mappings can successfully improve jumpstart and total reward. A set of experiments also shows how the algorithm can assist with source task selection by selecting the source task which is best able to minimize the offline prediction error. The primary contribution of MASTER is to demonstrate that autonomous transfer is possible, as the algorithm can learn inter-task mappings autonomously, which may then be used by any of the TL methods discussed in the previous section of this survey (Section 7).

In summary, this last section of the survey has discussed several methods able to learn inter-task mappings with different amounts of data. Although all make some assumptions about the amount of knowledge provided to the learner or the similarity between source and target tasks, these approaches represent an important step towards achieving fully autonomous transfer.

The methods in the section have been loosely ordered in terms of increasing autonomy. By learning inter-task mappings, these algorithms try to enable a TL agent to use past knowledge on a novel task without human intervention, even if the state variables or actions change. However, the question remains whether fully autonomous transfer would ever be useful in practice. Specifically, if there are no restrictions on the type of target task that could be encountered, why would one expect that past knowledge (a type of bias) would be useful when learning an encountered task, or even on the majority of tasks that could be encountered? This question is directly tied to the ability of TL algorithms to recognize when tasks are similar and when negative transfer may occur, both of which are discussed in more detail in the following section.

9. Open Questions

Although transfer learning in RL has made significant progress in recent years, there are still a number of open questions to be addressed. This section presents a selection of questions that we find particularly important. Section 9.1 discusses ways in which methods in the survey could potentially be extended and serves to highlight some of the methods most promising for future work. Section 9.2 then discusses the problem of negative transfer, currently one of the most troubling open questions. Lastly, Section 9.3 presents a set of possible research directions that the authors' believe will be most beneficial to the field of TL.

9.1 Potential Enhancements

One apparent gap in our taxonomy is a dearth of model-learning methods. Because model-learning algorithms are often more sample efficient than model-free algorithms, it is likely that TL will have a large impact on sample complexity when coupled with such efficient RL methods. Moreover, when a full model of the environment is learned in a source task, it may be possible for the target task learner to explicitly reason about how to refine or extend the model as it encounters disparities between it and the target task.

As mentioned in Section 5, transfer is an appealing way to set priors in a Bayesian setting. When in a MTL setting, it may be possible to accurately learn priors over a distribution of tasks, enabling a learner to better avoid negative transfer. One of the main benefits of transfer learning is the ability to bias learners so that they may find better solutions with less data; making these biases explicit through Bayesian priors may allow more efficient (and human-understandable) transfer methods. While there will likely be difficulties associated with scaling up current methods to handle complex tasks, possibly with a complex distribution hierarchy, it seems like Bayesian methods are particularly appropriate for transfer.

The idea of automatically modifying source tasks (cf., RTP Sherstov and Stone 2005, and suggested by Kuhlmann and Stone 2007) has not yet been widely adopted. However, such methods have the potential to improving transfer efficacy in settings where the target task learning performance is paramount. By developing methods that allow training on a sequence of automatically generated variations, TL agents may be able to train autonomously and gain experience that is exploitable in a novel task. Such an approach would be particularly relevant in the multi-task learning setting where the agent could leverage some assumptions about the distribution of the target task(s) it will see in the future.

None of the transfer methods in this survey are able to explicitly take advantage of any knowledge about changes in the reward function between tasks, and it may be particularly easy for humans to identify qualitative changes in reward functions. For example, if it was known that the target task rewards were twice that of the source task, it is possible that value-function methods may be able to automatically modify the source task value function with this background knowledge to enhance learning. As a second example, consider a pair of tasks where the goal state were moved from one edge of the state space to the opposite edge. While the learned transition information could be reused, the policy or value-function would need to be significantly altered to account for the new reward function. It is possible that inter-task mappings could be extended to account for changes in R between tasks, in addition to changes in A and in state variables.

Ideas from *theory revision* (Ginsberg, 1988) (also *theory refinement*) may help inform the automatic construction of inter-task mappings. For example, many methods initialize a target task agent to have Q-values similar to those in the source task agent. Transfer is likely to be successful (Taylor et al., 2007a) if the target task Q-values are close enough to the optimal Q-values that learning is improved, relative to not using transfer. There are also situations where a *syntactic* change to the knowledge would produce better transfer. For instance, if the target task's reward function were the inverse of the source task function, direct transfer of Q-values would be far from optimal. However, a TL algorithm that could recognize the inverse relationship may be able to use the source task knowledge more appropriately (such as initializing its behavior so that $\pi_{target}(s_{target}) \neq \pi_{source}(\chi_X(s_{target}))$.

Given a successful application of transfer, there are potentially two distinct benefits for the agent. First, transfer may help improve the agent's exploration so that it discovers higher-valued states more quickly. Secondly, transfer can help bias the agent's internal representation (e.g., its function approximator) so that it may learn faster. It will be important for future work to better distinguish between these two effects; decoupling the two contributions should allow for a better understanding of TL's benefits, as well as provide avenues for future improvements.

Of the thirty-four transfer methods discussed, only five (Tanaka and Yamamura, 2003; Sunmola and Wyatt, 2006; Ferguson and Mahadevan, 2006; Lazaric, 2008; Wilson et al., 2007) attempt to discover internal learning parameters (e.g., appropriate features or learning rate) so that future tasks in the same domain may be learned more efficiently. It is likely that other "meta-learning" methods could be useful. For instance, it may be possible to learn to use an appropriate function approximator, an advantageous learning rate, or even the most appropriate RL method. Although likely easier to accomplish in a MTL setting, such meta-learning may also be possible in transfer, given sufficiently strong assumptions about task similarity. Multiple heuristics regarding the best way to select RL methods and learning parameter settings for a particular domain exist, but typically such settings are chosen in an ad hoc manner. Transfer may be able to assist when setting such parameters, rather than relying on human intuition.

Section 8 discussed methods that learned an inter-task mapping, with the motivation that such a mapping could enable autonomous transfer. However, it is unclear if fully autonomous TL is realistic in an RL setting, or indeed is useful. In the majority of situations, a human will be somewhere in the loop and full autonomy is not necessary. Instead, it could be that mappings may be learned to *supplement* a human's intuition regarding appropriate mappings, or that a set of learned mappings could be proposed and then one selected by a human. It would be worthwhile to define realistic scenarios when fully autonomous transfer will be necessary, or to instead specify how (limited) human interaction will be coupled with mapping-learning methods.

Lastly, we hope that the idea of task-invariant knowledge will be extended. Rather than learning an appropriate representation across tasks, agent-space (Konidaris and Barto, 2007) and RRL techniques attempt to discover knowledge about the agent or the agent's actions which can be directly reused in novel tasks. The better techniques can successfully compartmentalize knowledge, separating what will usefully transfer and what will not will not, the easier it will be to achieve successful transfer without having to un-learn irrelevant biases.

9.2 Negative Transfer

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 fail according to one or more metrics. While we can say that it is possible to improve

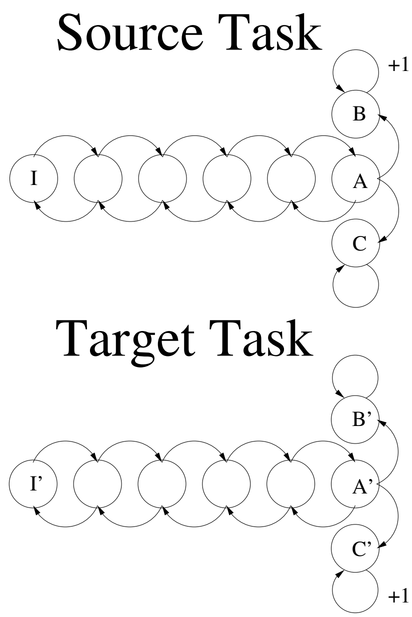

Figure 11: This figure depicts a pair of tasks that are likely to result in negative transfer for TL methods.

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.

Methods such as MASTER (Taylor et al., 2008c), which can measure task similarity via model prediction error, or region transfer (Lazaric, 2008), which examines the similarity of tasks at a local level rather than at a per-task level, can help assist when deciding if the agent should transfer or what the agent should transfer. However, neither method provides any theoretical guarantees about its effectiveness.

As an example of why it is difficult to define a metric for task similarity, consider the pair of tasks shown in Figure 11, which are extremely similar, but where direct transfer of a policy or action-value function will be detrimental. The source task in Figure 11 (top) is deterministic and discrete. The agent begins in state I and has one action available: `East`. Other states in the "hallway" have two applicable actions: `East` and `West`, except for state A, which also has the actions `North` and `South`. Once the agent executes `North` or `South` in state A, it will remain in state B or C (respectively) and continue self-transitioning. No transition has a reward, except for the self-transition in state B.

Now consider the target task in Figure 11 (bottom), which is the same as the source task, except that the self-transition from C$'$ is the only rewarded transition in the MDP. $Q^{\star}$(I$'$,`East`) in the target task (the optimal action-value function, evaluated at the state I$'$) is the same as $Q^{\star}$(I, `East`) in the source task. Indeed, the optimal policy in the target task differs at only a single state, A$'$, and the optimal action-value functions differ only at states A$'$, B$'$, and C$'$.

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*. In addition to this, or perhaps because of it, there are currently no methods for automatically *constructing* a source task given a target task.[11]

Homomorphisms (Ravindran and 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 and Singh, 2006). However, discovering homomorphisms is NP-hard (Ravindran and Barto, 2003a) 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.

9.3 New Directions

As suggested above, TL in RL domains is one area of machine learning where the empirical work has outpaced the theoretical. While there has been some work on the theory of transfer between classification tasks (cf., Baxter, 2000; Ben-David and Borbely, 2008), such analyses do not directly apply to RL settings. To our knowledge, there is only a single work analyzing the theoretical properties of transfer in RL (Phillips, 2006), where the authors use the Kantorovich and full models of two MDPs to calculate how well an optimal policy in one task will perform in a second task. Unfortunately, this calculation of policy performance may require more computation than directly learning in the target task. There is considerable room, and need for, more theoretical work in RL (cf., Bowling and Veloso, 1999). For example:

1. Provides guarantees about whether a particular source task can improve learning in a target task (given a particular type of knowledge transfer).

2. Correlates the amount of knowledge transferred (e.g., the number of samples) with the improvement in the source task.

3. Defines what an optimal inter-task mapping is, and demonstrates how transfer efficacy is impacted by the inter-task mapping used.

11. We distinguish this idea from Sherstov and Stone's 2005 approach. Their paper shows it is possible to construct source task perturbations and then allow an agent to spend time learning the set of tasks to attempt to improve learning on an (unknown) source task. Instead, it may be more effective to tailor a source task to a specific target task, effectively enabling an agent to reduce the total number of environmental interactions needed to learn.

The remainder of this section suggests other open areas.

Concept drift (Widmer and Kubat, 1996) in RL has not been directly addressed by any work in this survey. The idea of concept drift is related to a non-stationary environment: at certain points in time, the environment may change arbitrarily. As Ramon et al. (2007) note, "for transfer learning, it is usually known when the context change takes place. For concept drift, this change is usually unannounced." Current on-line learning methods may be capable of handling such changes by continually learning. However, it is likely that RL methods developed specifically to converge to a policy and then re-start learning when the concept changes will achieve higher performance, whether such drift is announced or unannounced.

Another question no work in this survey directly addresses is how to determine the optimal amount of source task training to minimize the target task training time or total training time. If the source task and target task were identical, the goal of reducing the target task training time would be trivial (by maximizing the source task training time) and the goal of minimizing total training time would be impossible. On the other hand, if the source task and target task were unrelated, it would be impossible to reduce the target task training time through transfer and the total training time would be minimized by not training in the source task at all. 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.

Transfer methods in this survey have used source task knowledge in many forms to better learn in a target task. However, none explicitly account for scaling differences between the two tasks. For instance, if a source task measured distance in meters and the target task measured distance in inches, constants would have to be updated manually rather than learned.

Another question not 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. One could imagine that a non-standard learning or exploration strategy may produce better transfer results, relative to standard strategies. 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 attempt to maximize the on-line reward accrued while learning) in non-transfer contexts (Şimşek and Barto, 2006).

Similarly, instead of always transferring information from the end of learning in the source task, an agent that knows its information will be used in a target task may decide to record information to transfer partway through training in the source task. For instance Taylor et al. (2007b) showed that transfer may be more effective when using policies trained for less time in the source task than when using those trained for more time. Although others have also observed similar behavior Mihalkova and Mooney (2008), the majority of work shows that increased performance in the source task is correlated with increased target task performance. Understanding how and why this effect occurs will help determine the most appropriate time to transfer information from one task to another.

We now present four possibilities for extending the current RL transfer work to different learning settings in which transfer has not been successfully applied.

- First, although two of the papers (Banerjee and Stone, 2007; Kuhlmann and Stone, 2007) in this survey have examined extensive games, none consider repeated normal form games or stochastic games (Shapley, 1953). For instance, one could consider learning how to play against a set of opponents so that when a new opponent is introduced, the learner may quickly adapt one of its previous strategies rather than completely re-learning a strategy. Another option would be for an agent to learn how to play one game and then transfer the knowledge to a different stochastic game. Due to similarities between RL and these two game playing settings, transfer methods described in this survey may be applied with relatively little modification.

- A second possibility for extending transfer is into the realm of partially observable MDPs (POMDPs). It may possible to learn a source POMDP and then use knowledge gained to heuristically speed up planning in a target POMDP. Additionally, because it is typically assumed that POMDP planners are given a complete and accurate model of a task, it may be possible to analytically compare source and target tasks before learning in order to determine if transfer would be beneficial, and if so, how best to use the past knowledge.

- Third, multi-agent MDP and POMDP learners may also be able to successfully exploit transfer. None of the work surveyed in this article focuses on explicit multi-agent learning (i.e., learning over the joint action space, or in an (adaptive) adversarial setting, as in Stone and Veloso 2000), but it is likely existing methods may be extended to the cooperative multi-agent setting. For instance, when formulating a problem as an MMDP or DEC-MDP, the agents must either reason over a joint action space or explicitly reason about how their actions affect others. It may be possible for agents to learn over a subset of actions first, and then gradually add actions (or joint actions) over time, similar to transferring between tasks with different action sets. The need for such speedups is particularly critical in distributed POMDPs, as solving them optimally as been shown to be NEXP-Complete (Bernstein et al., 2002). Transfer is one possible approach to making such problems more tractable, but to our knowledge, no such methods have yet been proposed.

- Fourth, as mentioned in Section 3.3, MTL methods in RL consider a sequence of tasks that are drawn sequentially from the same distribution. However, in supervised learning, multi-task learning typically involves learning multiple tasks *simultaneously*. There may be contexts in which an agent must learn multiple tasks concurrently, such as in hierarchical RL or when the agent has multiple reward functions or goals. Fully specifying such a scenario, and extending MTL methods to encompass this setting, could bring additional tools to RL researchers and help move TL in RL closer to TL in classification.

Lastly, in order to better evaluate TL methods, it would be helpful to have a standard set of domains and metrics. Ideally there would be a domain-independent metric for transfer learning, but it is unclear that such a metric can exist (see Section 2). Furthermore, it is unclear what *optimal transfer* would mean, but would likely depend on the scenario considered. Classification and regression have long benefited from standard metrics, such as precision and recall, and it is likely that progress in transfer will be likewise enhanced once standard metrics are agreed upon.

Standard test sets, such as the Machine Learning Repository at the University of California, Irvine (Asuncion and Newman, 2007), have also assisted the growth and progress of supervised learning, but there are currently no equivalents for RL. Furthermore, while there are some standard data sets for for transfer learning in classification,[12] none exist for transfer in RL. While there is

12. Found at `http://multitask.cs.berkeley.edu`.

some work in the RL community to standardize on a common interface and set of benchmark tasks (Tanner et al., 2008; Whiteson et al., 2008), no such standardization has been proposed for the transfer learning in RL community. Even in the absence of such a framework, we suggest that it is important for authors working in this area to:

- Clearly specify the setting: Is the source task learning time discounted? What assumptions are made about the relationship between the source target and target task?

- Evaluate the algorithm with a number of metrics: No one metric captures all possible benefits from transfer.

- Empirically or theoretically compare the performance of novel algorithms: To better evaluate novel algorithms, existing algorithms should be compared using standard metrics on a single task task.[13]

As discussed in Section 2.1, we do not think that TL for RL methods can be strictly ordered in terms of efficacy, due to the many possible goals of transfer. However, by standardizing on reporting methodology, TL algorithms can be more easily compared, making it easier to select an appropriate method in a given experimental setting.

Our hope is that TL questions, such as those presented in this section, will be addressed in the near future; our expectation is that transfer learning will become an increasingly powerful tool for the machine learning community.

Acknowledgments

We would like to thank Cynthia Matuszek and the anonymous reviewers for helpful comments and suggestions over multiple revisions. This work has taken place in 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), DARPA (FA8750-05-2-0283 and FA8650-08-C-7812), the Federal Highway Administration (DTFH61-07-H-00030), and General Motors.

References

Agnar Aamodt and Enric Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches, 1994.

Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML '05: Proceedings of the 22nd International Conference on Machine Learning*, pages 1–8, 2005.

13. One of the difficulties inherent in this proposal is that small variations in domain implementation may result in very different learning performances. While machine learning practitioners are able to report past results verbatim when using the same data set, many RL domains used in papers are not released. In order to compare with past work, RL researchers must reimplement, tune, and test past algorithms to compare with their algorithm on their domain implementation.

David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Proc. of the Eighteenth National Conference on Artificial Intelligence*, pages 119–125, 2002.

Andreas Argyrious, Theodoros Evgenion, and Massimiliano Pontil. Multitask reinforcement learning on the distribution of MDPs. *Machine Learning*, 2007.

Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *Proceedings of IAPR/IEEE Workshop on Visual Behaviors-1994*, pages 112–118, 1994.

Mehran Asadi and Manfred Huber. Effective control knowledge transfer through learning skill and representation hierarchies. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2054–2059, 2007.

Authur Asuncion and David J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Christopher G. Atkeson and Juan C. Santamaria. A comparison of direct and model-based reinforcement learning. In *Proceedings of the 1997 International Conference on Robotics and Automation*, 1997.

Bikramjit Banerjee and Peter Stone. General game learning using knowledge transfer. In *The 20th International Joint Conference on Artificial Intelligence*, pages 672–677, January 2007.

Bikramjit Banerjee, Yaxin Liu, and G. Michael Youngblood. ICML workshop on "Structural knowledge transfer for machine learning", June 2006.

Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Richard E. Bellman. A problem in the sequential design of experiments. *Sankhya*, 16:221–229, 1956.

Shai Ben-David and Reba Schuller Borbely. A notion of task relatedness yielding provable multiple-task learning guarantees. *Machine Learning*, 73:273–287, 2008.

Darrin C. Bentivegna, Christopher G. Atkeson, and Gordon Cheng. Learning from observation and practice using primitives. In *AAAI 2004 Fall Symposium on Real-life Reinforcement Learning*, October 2004.

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27 (4):819–840, November 2002.

Michael H. Bowling and Manuela M. Veloso. Bounding the suboptimality of reusing subproblem. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1340–1347, San Francisco, CA, USA, 1999.

James L. Carroll and Kevin Seppi. Task similarity measures for transfer in reinforcement learning task libraries. *Proceedings of 2005 IEEE International Joint Conference on Neural Networks*, 2: 803–808, 2005.

Rich Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in Neural Information Processing Systems 7*, pages 657–664, 1995.

Rich Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

Dongkyu Choi, Tolgo Konik, Negin Nejati, Chunki Park, and Pat Langley. Structural transfer of cognitive skills. In *Proceedings of the Eighth International Conference on Cognitive Modeling*, 2007.

William W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995.

Marco Colombetti and Marco Dorigo. Robot shaping: developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkeley, CA, 1993.

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

Tom Croonenborghs, Kurt Driessens, and Maurice Bruynooghe. Learning relational options for inductive transfer in relational reinforcement learning. In *Proceedings of the Seventeenth Conference on Inductive Logic Programming*, 2007.

DARPA. Transfer learning proposer information pamphlet, BAA #05-29, 2005.

Thomas Dean and Robert Givan. Model minimization in Markov decision processes. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 106–111, 1997.

Richard Dearden, Nir Friedman, and David Andre. Model based Bayesian exploration. In *Proceedings of the 1999 Conference on Uncertainty in Artificial Intelligence*, pages 150–159, 1999.

AArthur Dempster, Nan Laird, and Donald Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Soc. Set. B (methodological)*, 39:1–38, 1977.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Chris Drummond. Accelerating reinforcement learning by composing solutions of automatically identified subtasks. *Journal of Artificial Intelligence Research*, 16:59–104, 2002.

Saso Dzeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):5–52, April 2001.

Tom Erez and William D. Smart. What does shaping mean for computational reinforcement learning? In *Proceedings of the Seventh IEEE International Conference on Development and Learning*, pages 215–219, 2008.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Kimberly Ferguson and Sridhar Mahadevan. Proto-transfer learning in Markov decision processes using spectral methods. In *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, June 2006.

Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

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

Norm Ferns, Pablo Castro, Prakash Panangaden, and Doina Precup. Methods for computing state similarity in Markov decision processes. In *Proceedings of the 22nd Conference on Uncertainty in Artificial intelligence*, pages 174–181, 2006.

Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for Markov decision processes with infinite state spaces. In *Proceedings of the 2005 Conference on Uncertainty in Artificial Intelligence*, pages 201–208, 2005.

David Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49 (1/2):325–346, 2004.

Allen Ginsberg. Theory revision via prior operationalization. In *Proceedings of the 1988 National Conference on Artificial Intelligence*, pages 590–595, 1988.

Carlos Guestrin, Daphne Koller, Chris Gearhart, and Neal Kanodia. Generalizing plans to new environments in relational MDPs. In *International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, August 2003.

Okhtay Ilghami, Hector Munoz-Avila, Dana S. Nau, and David W. Aha. Learning approximate preconditions for methods in hierarchical plans. In *ICML '05: Proceedings of the 22nd International Conference on Machine learning*, pages 337–344, 2005.

Nicholas K. Jong and Peter Stone. Model-based exploration in continuous state spaces. In *The Seventh Symposium on Abstraction, Reformulation, and Approximation*, July 2007.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Zsolt Kalmár and Csaba Szepesvári. An evaluation criterion for macro learning and some results. Technical Report TR-99-01, Mindmaker Ltd., 1999.

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

W. Bradley Knox and Peter Stone. TAMER: training an agent manually via evaluative reinforcement. In *IEEE 7th International Conference on Development and Learning*, August 2008.

J. Zico Kolter, Pieter Abbeel, and Andrew Ng. Hierarchical apprenticeship learning with application to quadruped locomotion. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 769–776. MIT Press, Cambridge, MA, 2008.

George Konidaris and Andrew Barto. Autonomous shaping: knowledge transfer in reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 489–496, 2006.

George Konidaris and Andrew G. Barto. Building portable options: skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 895–900, 2007.

Gregory Kuhlmann and Peter Stone. Graph-based domain mapping for transfer learning in general games. In *Proceedings of The Eighteenth European Conference on Machine Learning*, September 2007.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in soar: the anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 1986.

Alessandro Lazaric. *Knowledge Transfer in Reinforcement Learning*. PhD thesis, Politecnico di Milano, 2008.

Bethany R. Leffler, Michael L. Littman, and Timothy Edmunds. Efficient reinforcement learning with relocatable action models. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 572–577, 2007.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.

Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 415–20, July 2006.

Richard Maclin and Jude W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3):251–281, 1996.

Richard Maclin, Jude Shavlik, Lisa Torrey, Trevor Walker, and Edward Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proceedings of the 20th National Conference on Artificial Intelligence*, 2005.

Michael G. Madden and Tom Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *Artificial Intelligence Review*, 21(3-4):375–398, 2004.

Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

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

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

Neville Mehta, Sriraam Natarajan, Prasad Tadepalli, and Alan Fern. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, 73(3):289–312, 2008.

Lilyana Mihalkova and Raymond J. Mooney. Transfer learning by mapping with minimal target data. In *Proceedings of the AAAI-08 Workshop on Transfer Learning for Complex Tasks*, July 2008.

Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

Andrew Moore. Variable resolution dynamic programming: efficiently learning action maps in multivariate real-valued state-spaces. In *Machine Learning: Proceedings of the Eighth International Conference*, June 1991.

Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, October 1993.

Andrew Y. Ng and Michael Jordan. PEGASUS: a policy search method for large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, 2000.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, 1999.

Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

Dirk Ormoneit and Saunak Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3): 161–178, 2002.

Theodore J. Perkins and Doina Precup. Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, The University of Massachusetts at Amherst, 1999.

Caitlin Phillips. Knowledge transfer in Markov decision processes. Technical report, McGill University, School of Computer Science, 2006. URL http://www.cs.mcgill.ca/~cphill/CDMP/summary.pdf.

Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Jan Ramon, Kurt Driessens, and Tom Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proceedings of The Eighteenth European Conference on Machine Learning*, September 2007.

Balaraman Ravindran and Andrew G. Barto. Model minimization in hierarchical reinforcement learning. In *Proceedings of the Fifth Symposium on Abstraction, Reformulation and Approximation*, 2002.

Balaraman Ravindran and Andrew G. Barto. An algebraic approach to abstraction in reinforcement learning. In *Proceedings of the Twelfth Yale Workshop on Adaptive and Learning Systems*, pages 109–114, 2003a.

Balaraman Ravindran and Andrew G. Barto. Relativized options: choosing the right transformation. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, pages 608–615, Menlo Park, CA, August 2003b. AAAI Press.

Daniel M. Roy and Leslie P. Kaelbling. Efficient Bayesian task-level transfer learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence, Hyderabad, India*, 2007.

Gavin Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University, 1994.

Manish Saggar, Thomas D'Silva, Nate Kohl, and Peter Stone. Autonomous learning of stable quadruped locomotion. In Gerhard Lakemeyer, Elizabeth Sklar, Domenico Sorenti, and Tomoichi Takahashi, editors, *RoboCup-2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Artificial Intelligence*, pages 98–109. Springer Verlag, Berlin, 2007.

Oliver G. Selfridge, Richard S. Sutton, and Andrew G. Barto. Training and tracking in robotics. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 670–672, 1985.

Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095–1100, October 1953.

Manu Sharma, Michael Holmes, Juan Santamaria, Arya Irani, Charles Isbell, and Ashwin Ram. Transfer learning in real-time strategy games using hybrid CBR/RL. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

Alexander A. Sherstov and Peter Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

Danny Silver, Goekhan Bakir, Kristin Bennett, Rich Caruana, Massimiliano Pontil, Stuart Russell, and Prasad Tadepalli. NIPS workshop on "Inductive transfer: 10 years later", December 2005.

Özgür Şimşek and Andrew G. Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning*, 2006.

Satinder Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

Satinder P. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339, 1992.

Burrhus F. Skinner. *Science and Human Behavior*. Colliler-Macmillian, 1953.

Vishal Soni and Satinder Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*, July 2006.

Ashwin Srinivasan. The aleph manual, 2001.

Peter Stone and Manuela Veloso. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

Funlade T. Sunmola and Jeremy L. Wyatt. Model transfer for Markov decision tasks via parameter matching. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)*, December 2006.

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

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

Richard S. Sutton, Anna Koop, and David Silver. On the role of tracking in stationary environments. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

Samarth Swarup and Sylvian R. Ray. Cross-domain knowledge transfer using structured representations. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*, July 2006.

Umar Syed and Robert Schapier. A multiplicative weights algorithm for apprenticeship learning. In *Advances in Neural Information Processing Systems 21*, 2007.

Erik Talvitie and Satinder Singh. An experts algorithm for transfer learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 2007.

Fumihide Tanaka and Masayuki Yamamura. Multitask reinforcement learning on the distribution of MDPs. *Transactions of the Institute of Electrical Engineers of Japan. C*, 123(5):1004–1011, 2003.

Brian Tanner, Adam White, and Richard S. Sutton. RL Glue and codecs, 2008. `http://mloss.org/software/view/151/`.

Matthew E. Taylor and Peter Stone. Representation transfer for reinforcement learning. In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*, November 2007a.

Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, June 2007b.

Matthew E. Taylor, Peter Stone, and Yaxin Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167, 2007a.

Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *The Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2007b.

Matthew E. Taylor, Alan Fern, Kurt Driessens, Peter Stone, Richard Maclin, and Jude Shavlik. AAAI workshop on "Transfer learning for complex tasks", July 2008a.

Matthew E. Taylor, Nicholas Jong, and Peter Stone. Transferring instances for model-based reinforcement learning. In *Proceedings of the Adaptive Learning Agents and Multi-Agent Systems (ALAMAS+ALAG) workshop at AAMAS-08*, May 2008b.

Matthew E. Taylor, Nicholas K. Jong, and Peter Stone. 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)*, pages 488–505, September 2008c.

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

Edward L. Thorndike and Robert S. Woodworth. The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review*, 8:247–261, 1901.

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

Sebastian Thrun and Lorien Pratt, editors. *Learning to learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

Lisa Torrey, Trevor Walker, Jude W. Shavlik, and Richard Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 412–424, 2005.

Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. Skill acquisition via transfer learning and advice taking. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 425–436, 2006.

Lisa Torrey, Jude W. Shavlik, Trevor Walker, and Richard Maclin. Relational macros for transfer in reinforcement learning. In *Proceedings of the Seventeenth Conference on Inductive Logic Programming*, 2007.

Thomas J. Walsh, Lihong Li, and Michael L. Littman. Transferring state abstractions between MDPs. In *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, June 2006.

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

Shimon Whiteson, Adam White, Brian Tanner, Richard S. Sutton Sutton, Doina Precup, Peter Stone, Michael Littman, Nikos Vlassis, and Martin Riedmiller. ICML workshop on "The 2008 RL-competition", July 2008.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical Bayesian approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, 2007.

Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.