Journal of Machine Learning Research 13 (2012) 1333-1371 Submitted 10/09; Revised 6/11; Published 5/12

Transfer in Reinforcement Learning via Shared Features

George Konidaris GDK@CSAIL.MIT.EDU
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
32 Vassar Street
Cambridge MA 02139

Ilya Scheidwasser SCHEIDWASSER.I@HUSKY.NEU.EDU
Department of Mathematics
Northeastern University
360 Huntington Avenue
Boston MA 02115

Andrew G. Barto BARTO@CS.UMASS.EDU
Department of Computer Science
University of Massachusetts Amherst
140 Governors Drive
Amherst MA 01003

Editor: Ronald Parr

Abstract

We present a framework for transfer in reinforcement learning based on the idea that related tasks share some common features, and that transfer can be achieved via those shared features. The framework attempts to capture the notion of tasks that are related but distinct, and provides some insight into when transfer can be usefully applied to a problem sequence and when it cannot. We apply the framework to the knowledge transfer problem, and show that an agent can learn a portable shaping function from experience in a sequence of tasks to significantly improve performance in a later related task, even given a very brief training period. We also apply the framework to skill transfer, to show that agents can learn portable skills across a sequence of tasks that significantly improve performance on later related tasks, approaching the performance of agents given perfectly learned problem-specific skills.

Keywords: reinforcement learning, transfer, shaping, skills

1. Introduction

One aspect of human problem-solving that remains poorly understood is the ability to appropriately generalize knowledge and skills learned in one task and apply them to improve performance in another. This effective use of prior experience is one of the reasons that humans are effective learners, and is therefore an aspect of human learning that we would like to replicate when designing machine learning algorithms.

Although reinforcement learning researchers study algorithms for improving task performance with experience, we do not yet understand how to effectively *transfer* learned skills and knowledge from one problem setting to another. It is not even clear which problem sequences allow transfer, which do not, and which do not need to. Although the idea behind transfer in reinforcement learning

seems intuitively clear, no definition or framework exists that usefully formalises the notion of "related but distinct" tasks—tasks that are similar enough to allow transfer but different enough to require it.

In this paper we present a framework for transfer in reinforcement learning based on the idea that related tasks share some common features and that transfer can take place through functions defined only over those shared features. The framework attempts to capture the notion of tasks that are related but distinct, and it provides some insight into when transfer can be usefully applied to a problem sequence and when it cannot. We then demonstrate the framework's use in producing algorithms for knowledge and skill transfer, and we empirically demonstrate the resulting performance benefits.

This paper proceeds as follows. Section 2 briefly introduces reinforcement learning, hierarchical reinforcement learning methods, and the notion of transfer. Section 3 introduces our framework for transfer, which is applied in Section 4 to transfer knowledge learned from earlier tasks to improve performance on later tasks, and in Section 5 to learn transferrable high-level skills. Section 7 discusses the implications and limitations of this work, and Section 8 concludes.

2. Background

The following sections briefly introduce the reinforcement learning problem, hierarchical reinforcement learning methods, and the transfer problem.

2.1 Reinforcement Learning

Reinforcement learning (Sutton and Barto, 1998) is a machine learning paradigm where an agent attempts to learn how to maximize a numerical reward signal over time in a given environment. As a reinforcement learning agent interacts with its environment, it receives a reward (or sometimes incurs a cost) for each action taken. The agent's goal is to use this information to learn to act so as to maximize the cumulative reward it receives over the future.

When the agent's environment is characterized by a finite number of distinct states, it is usually modeled as a finite Markov Decision Process (Puterman, 1994) described by a tuple $M = \langle S, A, P, R \rangle$, where S is the finite set of environment *states* that the agent may encounter; A is a finite set of *actions* that the agent may execute; $P(s'|s,a)$ is the probability of moving to state $s' \in S$ from state $s \in S$ given action $a \in A$; and R is a *reward function*, which given states s and s' and action a returns a scalar reward signal to the agent for executing action a in s and moving to s'.

The agent's objective is to maximize its cumulative reward. If the reward received by the agent at time k is denoted r_k, we denote this cumulative reward (termed *return*) from time t as $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$, where $0 < \gamma \leq 1$ is a *discount factor* that expresses the extent to which the agent prefers immediate reward over delayed reward.

Given a *policy* π mapping states to actions, a reinforcement learning agent may learn a *value function*, V, mapping states to expected return. If the agent is given or learns models of P and R, then it may update its policy as follows:

$$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')], \forall s \in S. \quad (1)$$

Once the agent has updated its policy, it must learn a new estimate of V. The repeated execution these two steps (value function learning and policy updates) is known as *policy iteration*. Under

certain conditions (Sutton and Barto, 1998), policy iteration is guaranteed to converge to an *optimal policy* π^* that maximizes return from every state. Policy iteration is usually performed implicitly: the agent simply defines its policy as Equation 1, effectively performing policy iteration after each value function update.

In some applications, states are described by vectors of real-valued features, making the state set a multidimensional continuous state space. (Hereafter we use the term *state space* to refer to both discrete state sets and continuous state spaces.) This creates two problems. First, one must find a way to compactly represent a value function defined on a multi-dimensional real-valued feature space. Second, that representation must facilitate *generalization*: in a continuous state space the agent may never encounter the same state twice and must instead generalize from experiences in nearby states when encountering a novel one.

The most common approximation scheme is *linear function approximation* (Sutton and Barto, 1998). Here, V is approximated by the weighted sum of a vector Φ of *basis functions*:

$$\bar{V}(\mathbf{s}) = \mathbf{w} \cdot \Phi(\mathbf{s}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{s}), \tag{2}$$

where ϕ_i is the ith basis function. Thus learning entails obtaining a weight vector $\mathbf{w}$ such that the weighted sum in Equation 2 accurately approximates V. Since $\bar{V}$ is linear in $\mathbf{w}$, when V's approximation as a weighted sum is not degenerate there is exactly one such optimal $\mathbf{w}$; however, we may represent complex value functions this way because each basis function ϕ_i may be an arbitrarily complex function of $\mathbf{s}$.

The most common family of reinforcement learning methods, and the methods used in this work, are *temporal difference methods* (Sutton and Barto, 1998). Temporal difference methods perform value function learning (and hence policy learning) online, through direct interaction with the environment. For more details see Sutton and Barto (1998).

2.2 Hierarchical Reinforcement Learning and the Options Framework

Much recent research has focused on hierarchical reinforcement learning (Barto and Mahadevan, 2003), where, apart from a given set of primitive actions, an agent can acquire and use higher-level macro actions built out of primitive actions. This paper adopts the options framework (Sutton et al., 1999) for hierarchical reinforcement learning; however, our approach could also be applied in other frameworks, for example the MAXQ (Dietterich, 2000) or Hierarchy of Abstract Machines (HAM) (Parr and Russell, 1997) formulations.

An option o consists of three components:

$$\begin{array}{lll} \pi_o: & (s,a) & \mapsto [0,1], \\ I_o: & s & \mapsto \{0,1\}, \\ \beta_o: & s & \mapsto [0,1], \end{array}$$

where π_o is the *option policy* (which describes the probability of the agent executing action a in state s, for all states in which the option is defined), I_o is the *initiation set* indicator function, which is 1 for states where the option can be executed and 0 elsewhere, and β_o is the *termination condition*, giving the probability of the option terminating in each state (Sutton et al., 1999). The options framework provides methods for learning and planning using options as temporally extended actions in the standard reinforcement learning framework (Sutton and Barto, 1998).

Algorithms for learning new options must include a method for determining when to create an option or alter its initiation set, how to define its termination condition, and how to learn its policy. Policy learning is usually performed by an off-policy reinforcement learning algorithm so that the agent can update many options simultaneously after taking an action (Sutton et al., 1998).

Creation and termination are usually performed by the identification of goal states, with an option created to reach a goal state and terminate when it does so. The initiation set is then the set of states from which the goal is reachable. Previous research has selected goal states by a variety of methods, for example: visit frequency and reward gradient (Digney, 1998), visit frequency on successful trajectories (McGovern and Barto, 2001), variable change frequency (Hengst, 2002), relative novelty (Şimşek and Barto, 2004), clustering algorithms and value gradients (Mannor et al., 2004), local graph partitioning (Şimşek et al., 2005), salience (Singh et al., 2004), causal decomposition (Jonsson and Barto, 2005), and causal analysis of expert trajectories (Mehta et al., 2008). Other research has focused on extracting options by exploiting commonalities in collections of policies over a single state space (Thrun and Schwartz, 1995; Bernstein, 1999; Perkins and Precup, 1999; Pickett and Barto, 2002).

2.3 Transfer

Consider an agent solving a sequence of n problems, in the form of a sequence of Markov Decision Processes $M_1, ..., M_n$. If these problems are somehow "related", and the agent has solved problems $M_1, ..., M_{n-1}$, then it seems intuitively reasonable that the agent should be able to use knowledge gained in their solutions to solve M_n faster than it would be able to otherwise. The transfer problem is the problem of how to obtain, represent and apply such knowledge.

Since transfer hinges on the tasks being related, the nature of that relationship will define how transfer can take place. For example, it is common to assume that all of the tasks have the same state space, action set and transition probabilities but differing reward functions, so that for any i, $M_i = \langle S, A, P, R_i \rangle$. In that case, skills learned in the state space and knowledge about the structure of the state space from previous tasks can be transferred, but knowledge about the optimal policy cannot.

In many transfer settings, however, each task in the sequence has a distinct state space, but the tasks nevertheless seem intuitively related. In the next section, we introduce a framework for describing the commonalities between tasks that have different state spaces and action sets.

3. Related Tasks Share Common Features

Successful transfer requires an agent that must solve a sequence of tasks that are related but distinct—different, but not so different that experience in one is irrelevant to experience in another. How can we define such a sequence? How can we use such a definition to perform transfer?

Consider the illustrative example of an indoor mobile robot required to perform many learning tasks over its lifetime. Although the robot might be equipped with a very rich set of sensors—for example, laser range finders, temperature and pressure gauges—when facing a particular task it will construct a task-specific representation that captures that task's essential features. Such a task-specific representation ensures that the resulting learning task is no more difficult than necessary, and depends on the complexity of the problem rather than the robot (adding more sensors or actuators should not make an easy task hard). In the reinforcement learning setting, a plausible design for such a robot would use a task-specific MDP, most likely designed to be as small as possible

(without discarding information necessary for a solution), and discrete (so that the task does not require function approximation).

Thus, a robot given the tasks of searching for a particular type of object in two different buildings B_1 and B_2 might form two completely distinct discrete MDPs, M_1 and M_2, most likely as topological maps of the two buildings. Then even though the robot should be able to share information between the two problems, without further knowledge there is no way to transfer information between them *based only on their description as MDPs*, because the state labels and transition probabilities of M_1 and M_2 need have no relation at all.

We argue that finding relationships between pairs of arbitrary MDPs is both unnecessarily difficult and misses the connection between these problems. The problems that such a robot might encounter are all related *because they are faced by the same agent*, and therefore the same sensor features are present in each, even if those shared features are abstracted away when the problems are framed as MDPs. If the robot is seeking a heat-emitting object in both B_1 and B_2, it should be able to learn after solving B_1 that its temperature gauge is a good predictor of the object's location, and use it to better search B_2, even though its temperature gauge reading does not appear as a feature in either MDP.

When trying to solve a single problem, we aim to create a minimal problem-specific representation. When trying to transfer information across a sequence of problems, we should instead concentrate on what is common across the sequence. We therefore propose that *what makes tasks related is the existence of a feature set that is shared and retains the same semantics across tasks.* To define what we mean by a feature having the same semantics across tasks, we define the notion of a *sensor*.

Consider a parametrized class of tasks $\Gamma(\theta)$, where Γ returns a task instance given parameter $\theta \in \Theta$. For example, Γ might be the class of square gridworlds, and θ might fix obstacle and goal locations and size. We can obtain a sequence of tasks $M_1, ..., M_n$ via a sequence of task parameters $\theta_1, ..., \theta_n$.

Definition 1 *A sensor ξ is a function mapping a task instance parameter $\theta \in \Theta$ and state $s_\theta \in S_\Theta$ of the task obtained using θ to a real number f:*

$$\xi : (\theta, s_\theta) \mapsto f.$$

The important property of ξ is that it is a function defined over all tasks in Γ: it produces a feature, f, that describes some property of an environment given that environment's parameters and current state. For example, f might describe the distance from a robot in a building to the nearest wall; this requires both the position of the robot in the building (the problem state) and the layout of the building (the environment parameters). The feature f has the same semantics across tasks because it is generated by the same function in each task instance.[1]

An agent may in general be equipped with a suite of such sensors, from which it can read at any point to obtain a feature vector. We call the space generated by the resulting features an *agent-space*, because it is a property of the agent rather than any of the tasks individually, as opposed to the problem-specific state space used to solve each problem (which we call a *problem-space*).

We note that in some cases the agent-space and problem-spaces used for a sequence of tasks may be related, for example, each problem-space might be formed by appending a task-specific amount

1. We exclude degenerate cases, for example where ξ simply uses θ as an index and produces completely different outputs for different values of θ, or where ξ returns a constant, or completely random, value.

of memory to agent-space. However, in general it may not be possible to recover an agent-space descriptor from a problem-space descriptor, or vice versa. The functions mapping the environment to each descriptor are distinct and must be designed (or learned outside of the reinforcement learning process) with different objectives.

We now model each problem in a sequence as an MDP augmented with an agent-space, writing the augmented MDP corresponding to the ith problem as:

$$M_i = \langle S_i, A_i, P_i, R_i, D \rangle,$$

where D (the agent-space) is a feature space defined across all tasks. For any state in any of the environments, the agent also obtains an observation (or *descriptor*) $d \in D$, the features of which have the same semantics across all tasks.

The core idea of our framework is that task learning occurs in problem-space, and transfer can occur via agent-space. If we have an MDP augmented with features that are known to be shared, we can use those shared features as a bridge across which knowledge can be transferred. This leads to the following definition:

Definition 2 *A sequence of tasks is* related *if that sequence has a non-empty agent-space—that is, if a set of shared features exist in all of the tasks.*

A further definition will prove useful in understanding when the transfer of information about the value function is useful:

Definition 3 *We define a sequence of related tasks to be* reward-linked *if the reward function for all tasks is the same sensor, so that rewards are allocated the same way for all tasks (for example, reward is always x for finding food).*

A sequence of tasks must be (at least approximately) reward-linked if we aim to transfer information about the optimal value function: if the reward functions in two tasks use different sensors then there is no reason to hope that their value functions contain useful information about each other.

If a sequence of tasks is related, we may be able to perform effective transfer by taking advantage of the shared space. If no such space exists, we cannot transfer across the sequence because there is no view (however abstract or lossy) in which the tasks share common features. If we can find an agent-space that is also usable as a problem-space for every task in the sequence, then we can treat the sequence as a set of tasks in the same space (by using D directly as a state space) and perform transfer directly by learning about the structure of this space. If in addition the sequence is reward-linked, then the tasks are not distinct and transfer is trivial because we can view them as a single problem. However, there may be cases where a shared problem-space exists but results in slow learning, and using task-specific problem-spaces coupled with a transfer mechanism is more practical.

We can therefore define the working hypothesis of this paper as follows:

> We can usefully describe two tasks as related when they share a common feature space, which we term an *agent-space*. If learning to solve each individual task is possible and feasible in agent-space, then transfer is trivial: the tasks are effectively a single task and we can learn a single policy in agent-space for all tasks. If it is not, then transfer between two tasks can nevertheless be effected through agent-space, either through the

transfer of knowledge about the value function (when the tasks are reward-linked), or through the transfer of skills defined in agent-space.

In the following sections we use this framework to build agents that perform these two different types of transfer. Section 4 shows that an agent can transfer value-functions learned in agent-space to significantly decrease the time taken to find an initial solution to a task, given experience in a sequence of related and reward-linked tasks. In Section 5 we show that an agent can learn portable high-level skills directly in agent-space which can dramatically improve task performance, given experience in a sequence of related tasks.

4. Knowledge Transfer

In this section, we show that agents that must repeatedly solve the same type of task (in the form of a sequence of related, reward-linked tasks) can transfer useful knowledge in the form of a *portable shaping function* that acts as an initial value function and thereby endows the agent with an initial policy. This significantly improves initial performance in later tasks, resulting in agents that can, for example, learn to solve difficult tasks quickly after being given a set of relatively easy training tasks.

We empirically demonstrate the effects of knowledge transfer using a relatively simple demonstration domain (a rod positioning task with an artificial agent space) and a more challenging domain (Keepaway). We argue (in Section 4.5) that this has the effect of creating agents which can learn their own heuristic functions.

4.1 Shaping

Shaping is a popular method for speeding up reinforcement learning in general, and goal-directed exploration in particular (Dorigo and Colombetti, 1998). Although this term has been applied to a variety of different methods within the reinforcement learning community, only two are relevant here. The first is the gradual increase in complexity of a single task toward some given final level (for example, Randløv and Alstrøm 1998; Selfridge et al. 1985), so that the agent can safely learn easier versions of the same task and use the resulting policy to speed learning as the task becomes more complex.[2] Unfortunately, this type of shaping does not generally transfer between tasks—it can only be used to gently introduce an agent to a single task, and is therefore not suited to a sequence of distinct tasks.

Alternatively, the agent's reward function could be augmented through the use of intermediate shaping rewards or "progress indicators" (Matarić, 1997) that provide an augmented (and hopefully more informative) reinforcement signal to the agent. This has the effect of shortening the reward horizon of the problem—the number of correct actions the agent must execute before receiving a useful reward signal (Laud and DeJong, 2003). Ng et al. (1999) proved that an arbitrary externally specified reward function could be included as a potential-based shaping function in a reinforcement learning system without modifying its optimal policy. Wiewiora (2003) showed that this is equivalent to using the same reward function as a non-uniform initial state value function, or with

2. We note that this definition of shaping is closest to its original meaning in the psychology literature, where it refers to a process by which an experimenter rewards an animal for behavior that progresses toward the completion of a complex task, and thereby guides the animal's learning process. As such it refers to a training technique, not a learning mechanism (see Skinner, 1938).

a small change in action selection, as an initial state-action value function (Wiewiora et al., 2003). Thus, we can use any function we like as an initial value function for the agent, even if (as is often the case in function approximation) it is not possible to directly initialize the value function. The major drawback is that designing such a shaping function requires significant engineering effort. In the following sections we show that an agent can learn its own shaping function from experience across several related, reward-linked tasks without having it specified in advance.

4.2 Learning Portable Shaping Functions

As before, consider an agent solving n problems with MDPs $M_1,...,M_n$, each with their own state space, denoted $S_1,...,S_n$ and augmented with agent-space features. We associate a four-tuple σ_i^j with the ith state in M_j:

$$\sigma_i^j = \langle s_i^j, d_i^j, r_i^j, v_i^j \rangle,$$

where s_i^j is the usual problem-space state descriptor (sufficient to distinguish this state from the others in S_j), d_i^j is the agent-space descriptor, r_i^j is the reward obtained at the state and v_i^j is the state's value (expected total reward for action starting from the state). The goal of value-function based reinforcement learning is to obtain the v_i^j values for each state in the form of a value function V_j:

$$V_j : s_i^j \mapsto v_i^j.$$

V_j maps problem-specific state descriptors to expected return, but it is not portable between tasks, because the form and meaning of s_i^j (as a problem-space descriptor) may change from one task to another. However, the form and meaning of d_i^j (as an agent-space descriptor) does not change. Since we want an estimator of return that is portable across tasks, we introduce a new function L that is similar to each V_j, but that estimates expected return given an agent-space descriptor:

$$L : d_i^j \mapsto v_i^j.$$

L is also a value function, but it is defined over portable agent-space descriptors rather than problem-specific state space descriptors. As such, we could consider it a form of feature-based value function approximation and update it online (using a suitable reinforcement learning algorithm) during each task. Alternatively, once an agent has completed some task S_j and has learned a good approximation of the value of each state using V_j, it can use its (d_i^j, v_i^j) pairs as training examples for a supervised learning algorithm to learn L. Since L is portable, we can in addition use samples from multiple related, reward-linked tasks.

After a reasonable amount of training, L can be used to estimate a value for newly observed states in any future related and reward-linked tasks. Thus, when facing a new task M_k, the agent can use L to provide a good initial estimate for V_k that can be refined using a standard reinforcement learning algorithm. Alternatively (and equivalently), L could be used as an external shaping reward function.

4.3 A Rod Positioning Experiment

In this section we empirically evaluate the potential benefits of a learned shaping function in a rod positioning task (Moore and Atkeson, 1993), where we add a simple artificial agent space that can be easily manipulated for experimental purposes.

Each task consists of a square workspace that contains a rod, some obstacles, and a target. The agent is required to maneuver the rod so that its tip touches the target (by moving its base coordinate or its angle of orientation) while avoiding obstacles. An example 20x20 unit task and solution path are shown in Figure 1.

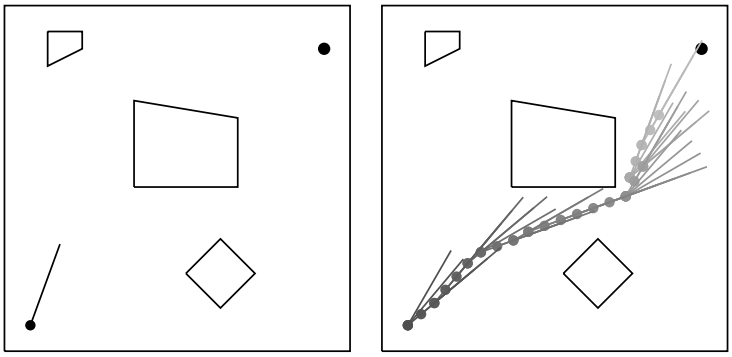


Figure 1: A 20x20 rod positioning task and one possible solution path.

Following Moore and Atkeson (1993), we discretize the state space into unit x and y coordinates and 10° angle increments. Thus, each state in the problem-space can be described by two coordinates and one angle, and the actions available to the agent are movement of one unit in either direction along the rod's axis, or a 10^o rotation in either direction. If a movement causes the rod to collide with an obstacle, it results in no change in state, so the portions of the state space where any part of the rod would be interior to an obstacle are not reachable. The agent receives a reward of -1 for each action, and a reward of 1000 when reaching the goal (whereupon the current episode ends).

We augment the task environment with five beacons, each of which emits a signal that drops off with the square of the Euclidean distance from a strength of 1 at the beacon to 0 at a distance of 60 units. The tip of the rod has a sensor array that can detect the values of each of these signals separately at the adjacent state in each action direction. Since these beacons are present in every task, the sensor readings are an agent-space and we include an element in the agent that learns L and uses it to predict reward for each adjacent state given the five signal levels present there.

The usefulness of L as a reward predictor will depend on the relationship between beacon placement and reward across a sequence of individual rod positioning tasks. Thus we can consider the beacons as simple abstract signals present in the environment, and by manipulating their placement (and therefore their relationship to reward) across the sequence of tasks, we can experimentally evaluate the usefulness of various forms of L.

4.3.1 EXPERIMENTAL STRUCTURE

In each experiment, the agent is exposed to a sequence of training experiences, during which it is allowed to update L. After each training experience, it is evaluated in a large test case, during which it is *not* allowed to update L.

Each individual training experience places the agent in a small task, randomly selected from a randomly generated set of 100 such tasks, where it is given sufficient time to learn a good solution. Once this time is up, the agent updates L using the value of each visited state and the sensory signal

present at it, before it is tested on the much larger test task. All state value tables are cleared between training episodes.

Each agent performed reinforcement learning using Sarsa(λ) ($\lambda = 0.9, \alpha = 0.1, \gamma = 0.99$, $\varepsilon = 0.01$) in problem-space and used training tasks that were either 10x10 (where it was given 100 episodes to converge in each training task), or 15x15 (when it was given 150 episodes to converge), and tested in a 40x40 task.[3] L was a linear estimator of reward, using either the five beacon signal levels and a constant as features (requiring 6 parameters, and referred to as the linear model) or using those with five additional features for the square of each beacon value (requiring 11 parameters, referred to as the quadratic model). All parameters were initialized to 0, and learning for L was accomplished using gradient descent with $\alpha = 0.001$. We used two experiments with different beacon placement schemes.

4.3.2 FOLLOWING A HOMING BEACON

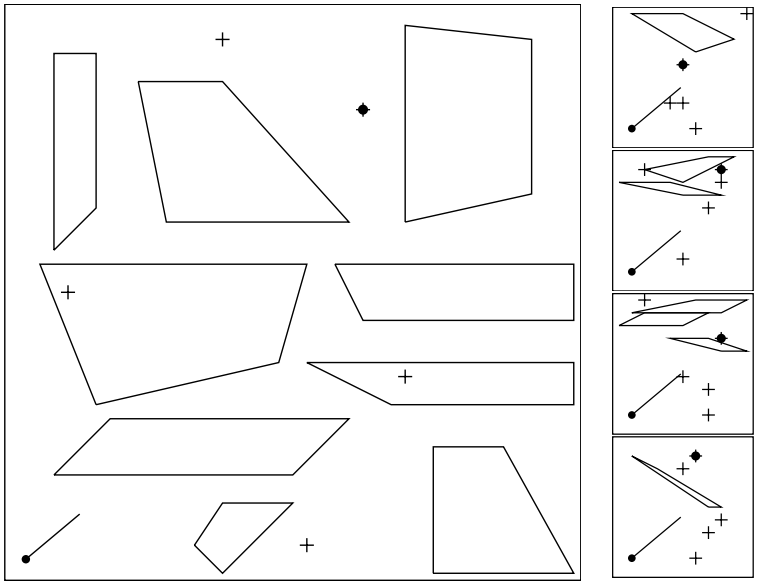
In the first experiment, we always placed the first beacon at the target location, and randomly distributed the remainder throughout the workspace. Thus a high signal level from the first beacon predicts high reward, and the others should be ignored. This is a very informative indication of reward that should be easy to learn, and can be well approximated even with a linear L. Figure 2 shows the 40x40 test task used to evaluate the performance of each agent, and four sample 10x10 training tasks.

Figure 3(a) shows the number of steps (averaged over 50 runs) required to first reach the goal in the test task, against the number of training tasks completed by the agent for the four types of learned shaping elements (linear and quadratic L, and either 10x10 or 15x15 training tasks). It also shows the average number of steps required by an agent with a uniform initial value of 0 (agents with a uniform initial value of 500 performed similarly while first finding the goal). Note that there is just a single data point for the uniform initial value agents (in the upper left corner) because their performance does not vary with the number of training experiences.

Figure 3(a) shows that training significantly lowers the number of steps required to initially find the goal in the test task in all cases, reducing it after one training experience from over $100,000$ steps to at most just over $70,000$, and by six episodes to between $20,000$ and $40,000$ steps. This difference is statistically significant (by a t-test, $p < 0.01$) for all combinations of L and training task sizes, even after just a single training experience. Figure 3(a) also shows that the complexity of L does not appear to make a significant difference to the long-term benefit of training (probably because of the simplicity of the reward indicator), but training task size does. The difference between the number of steps required to first find the goal for 10x10 and 15x15 training task sizes is statistically significant ($p < 0.01$) after 20 training experiences for both linear and quadratic forms of L, although this difference is clearer for the quadratic form, where it is significant after 6 training experiences.

Figure 3(b) shows the number of steps (averaged over 50 runs) required to reach the goal as the agents repeat episodes in the test task, after completing 20 training experiences (note that L is never updated in the test task), compared to the number of steps required by agents with value tables uniformly initialized to 0 and 500. This illustrates the difference in overall learning performance on a single new task between agents that have had many training experiences and agents that have

3. We note that in general the tasks used to train the agent need not be smaller than the task used to test it. We used small training tasks in this experiment to highlight the fact that the size of problem-space may differ between related tasks.



a b

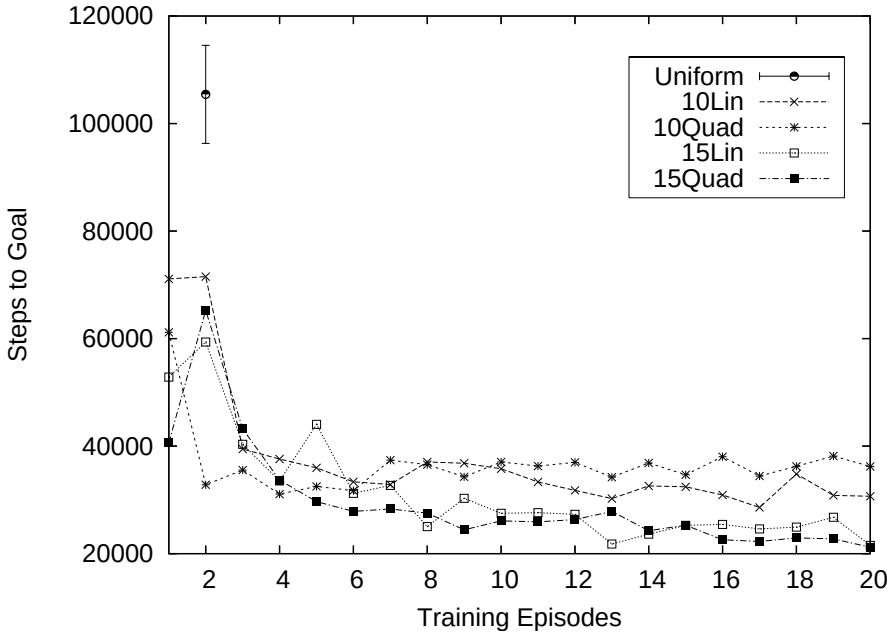
Figure 2: The homing experiment 40x40 test task (a) and four sample 10x10 training tasks (b). Beacon locations are shown as crosses, and the goal is shown as a large dot. Note that the first beacon is on the target in each task. The optimal solution for the test task requires 69 steps.

not. Figure 3(b) shows that the learned shaping function significantly improves performance during the first few episodes of learning, as expected. It also shows that the number of episodes required for convergence is roughly the same as that of an agent using a uniformly optimistic value table initialization of 500, and slightly longer than that of an agent using a uniformly pessimistic value table initialization of 0. This suggests that once a solution is found the agent must "unlearn" some of its overly optimistic estimates to achieve convergence. Note that a uniform initial value of 0 works well here because it discourages extended exploration, which is unnecessary in this domain.

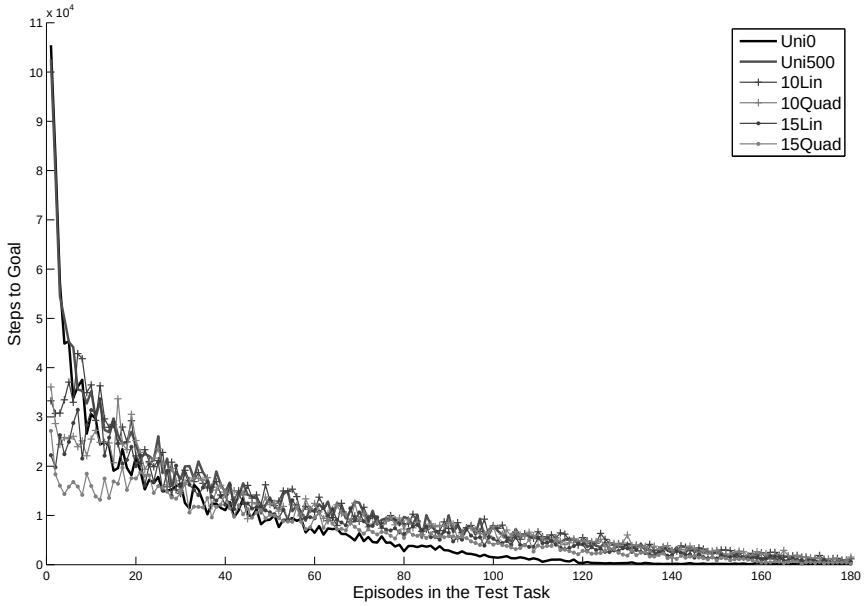
4.3.3 FINDING THE CENTER OF A BEACON TRIANGLE

In the second experiment, we arranged the first three beacons in a triangle at the edges of the task workspace, so that the first beacon lay to the left of the target, the second directly above it, and the third to its right. The remaining two were randomly distributed throughout the workspace. This provides a more informative signal, but results in a shaping function that is harder to learn. Figure 4 shows the 10x10 sample training tasks given in Figure 2 after modification for the triangle experiment. The test task was similarly modified.

Figure 5(a) shows the number of steps initially required to reach the goal for the triangle experiment, again showing that completing even a single training task results in a statistically significant ($p < 0.01$ in all cases) reduction from the number required by an agent using uniform initial values, from just over $100,000$ steps to at most just over $25,000$ steps after a single training episode. Figure



(a) The average number of steps required to first reach the goal in the homing test task, for agents that have completed varying numbers of training task episodes.



(b) Steps to reward against episodes in the homing test task for agents that have completed 20 training tasks.

Figure 3: Results for the homing task.

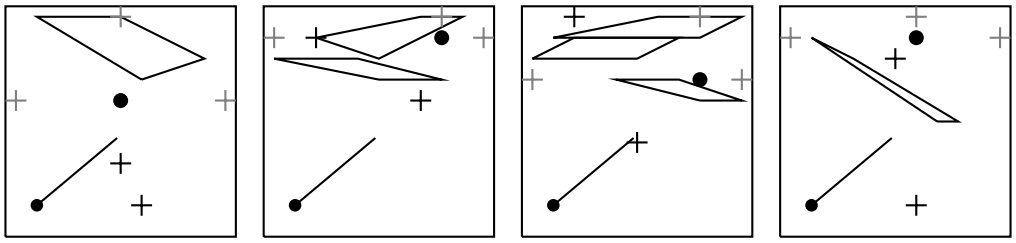


Figure 4: Sample 10x10 training tasks for the triangle experiment. The three beacons surrounding the goal in a triangle are shown in gray.

5(a) also shows that there is no significant difference between forms of L and size of training task. This suggests that extra information in the agent-space more than makes up for a shaping function being difficult to accurately represent—in all cases the performance of agents learning using the triangle beacon arrangement is better than that of those learning using the homing beacon arrangement. Figure 5(b) shows again that the initial few episodes of repeated learning in the test task are much faster, and again that the total number of episodes required to converge lies somewhere between the number required by an agent initializing its value table pessimistically to 0 and one initializing it optimistically to 500.

4.3.4 SENSITIVITY ANALYSIS

So far, we have used shared features that are accurate in the sense that they provide a signal that is uncorrupted by noise and that has exactly the same semantics across tasks. In this section, we empirically examine how sensitive a learned shaping reward might be to the presence of noise, both in the features and in their role across tasks.

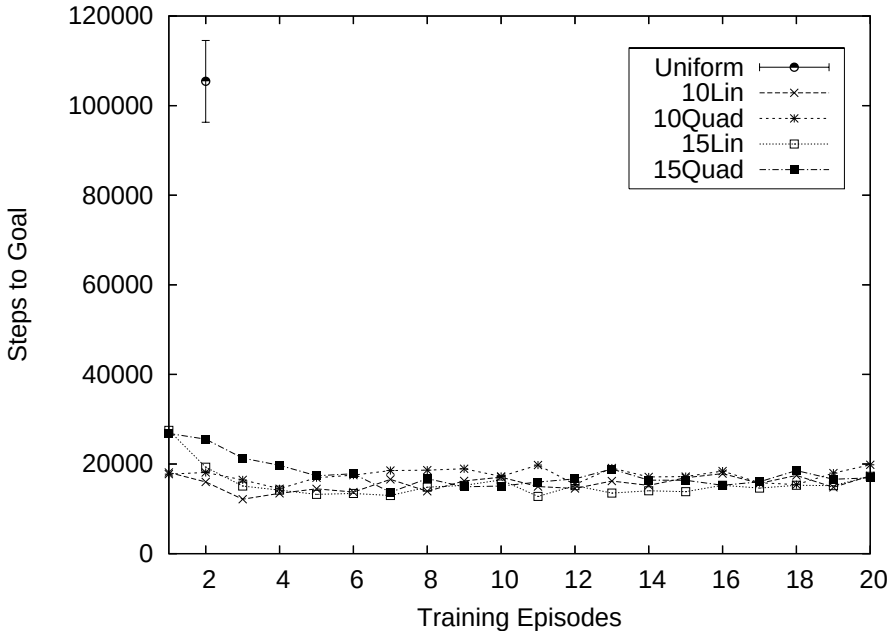
To do so, we repeat the above experiments (using training tasks of size 15, and a quadratic approximator) but with only a single beacon whose position is given by the following formula:

$$\mathbf{b} = (1-\eta)\mathbf{g} + \eta\mathbf{r},$$

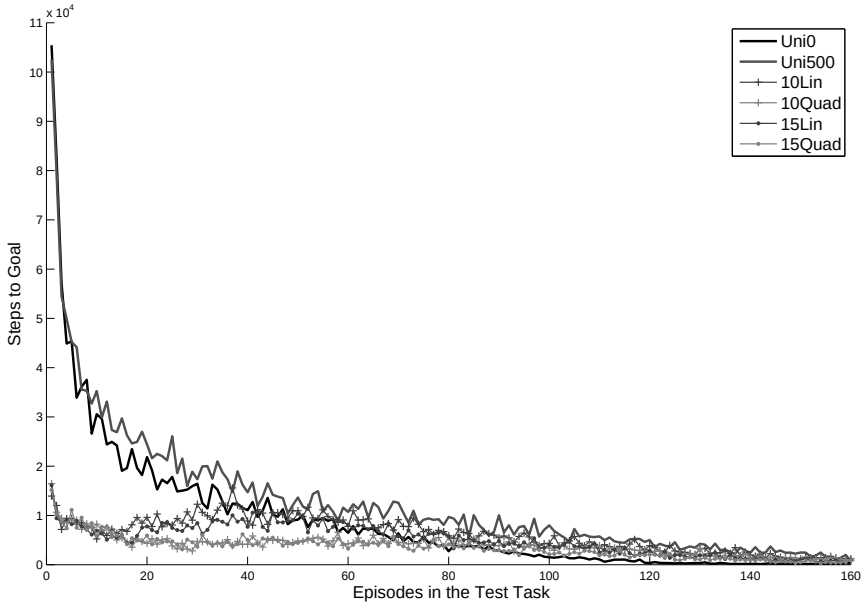
where $\mathbf{g}$ is the coordinate vector of the target, $\eta \in [0,1]$ is a noise parameter, and $\mathbf{r}$ is a co-ordinate vector generated uniformly at random. Thus, when $\eta = 0$ we have no noise and the beacon is always placed directly over the goal; when $\eta = 1$, the beacon is placed randomly in the environment. Varying η between 0 and 1 allows us to manipulate the amount of noise present in the beacon's placement, and hence in the shared feature used to learn a portable shaping function. We consider two scenarios.

In the first scenario, the same η value is used to place the beacon in both the training and the test problem. This corresponds to a signal that is perturbed by noise, but whose semantics remain the same in both source and target tasks. This measures how sensitive learned shaping rewards are to feature noise, and so we call this the *noisy-signal* task. The results are shown in Figure 6(a) and 6(b).

Figure 6(a) measures the number of steps required to complete the first episode in the large test problem, given experience in various numbers of training problems and varying levels of noise. The results show that transfer is fairly robust to noise, resulting in an improvement over starting from

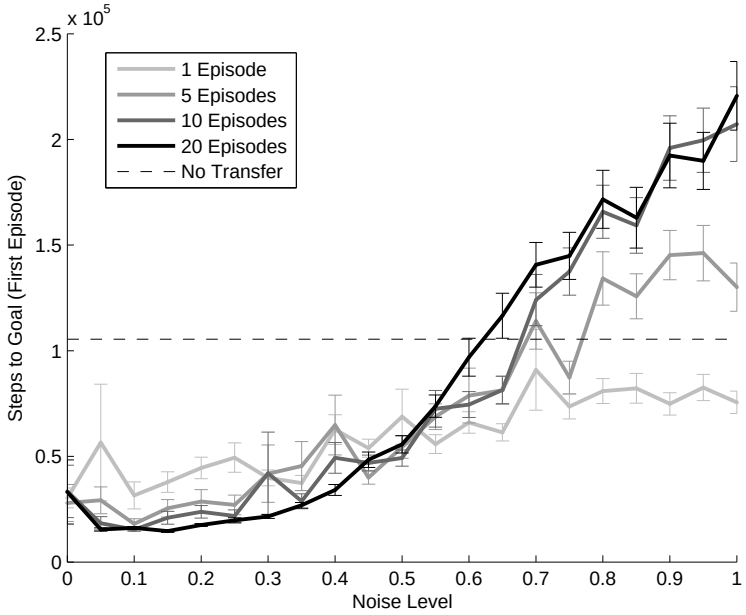


(a) The average number of steps required to first reach the goal in the triangle test task, for agents that have completed varying number of training task episodes.

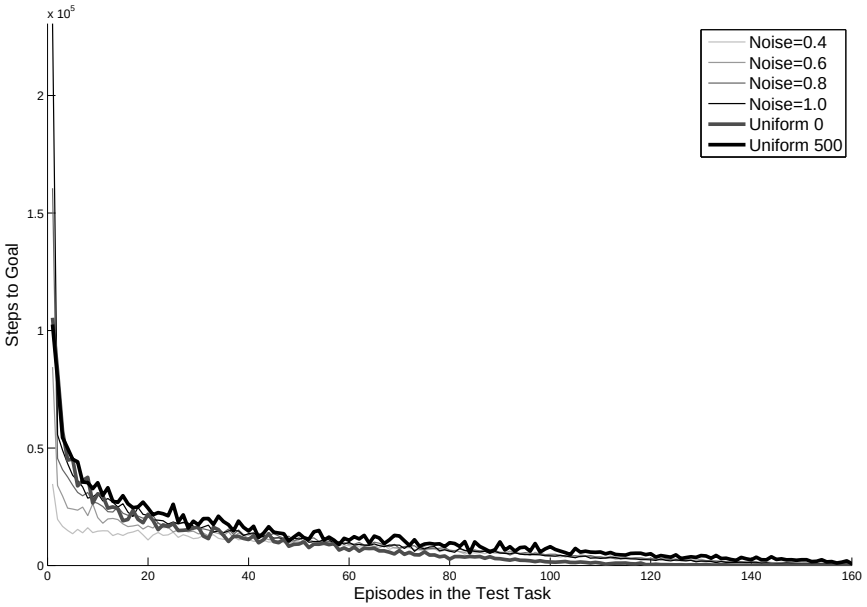


(b) Steps to reward against episodes in the triangle test task for agents that have completed 20 training tasks.

Figure 5: Results for the triangle task.



(a) The average number of steps required to first reach the test task goal given a predictor learned using a noisy signal.



(b) Steps to reward against episodes in the test task for agents that have completed 20 training task episodes using a noisy signal.

Figure 6: Results for the *noisy-signal* task.

scratch that drops with increased noise but still does better until $\eta > 0.6$, when the feature has more noise than signal.

Higher levels of noise more severely affects agents that have seen higher numbers of training problems, until a performance floor is reached between 5 and 10 training problems. This reflects the training procedure used to learn L, whereby each training problem results in a small adjustment of L's parameters and those adjustments accumulate over several training episodes.

Similarly, Figure 6(b) shows learning curves in the test problem for agents that have experienced 20 test problems, with varying amounts of noise. We see that, although these agents often do worse than learning from scratch in the first episode, they subsequently do better when $\eta < 1$, and again converge at roughly the same rate as agents that use an optimistic initial value function.

In the second scenario, η is zero in the training problems, but non-zero in the test problem. This corresponds to a feature which has slightly different semantics in the source and target tasks, and thus measures how learning is affected by an imperfect or approximate choice of agent space features. We call this the *noisy-semantics* task.

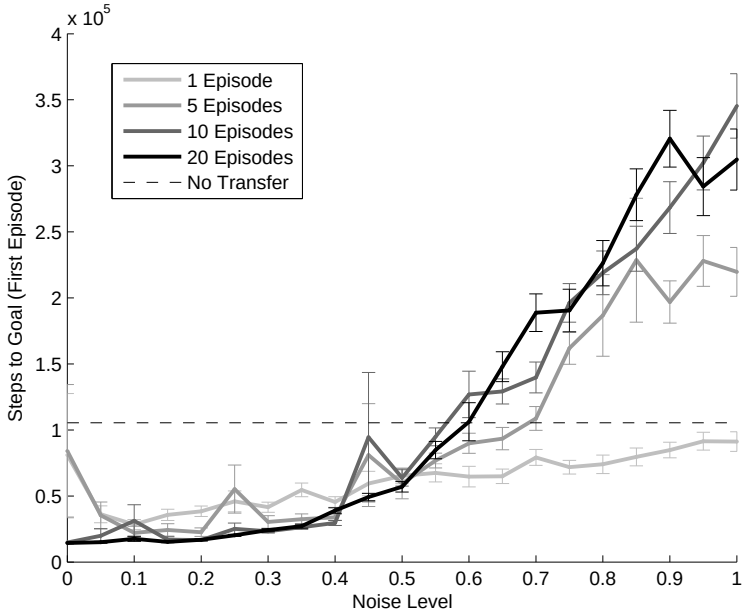
Results for the noisy-semantics task are given in Figures 7(a) and 7(b). These two graphs show that transfer achieves a performance benefit when $\eta < 0.5$—when there is at least as much signal as noise—and the more training problems the agent has solved, the worse its performance will be when $\eta = 1$. However, the possible performance penalty for high η is more severe—an agent using a learned shaping function that rewards it for following a beacon signal may take nearly four times as long to first solve the test problem when that feature becomes random (at $\eta = 1$). Again, however, when $\eta < 1$ the agents recover after their first episode to outperform agents that learn from scratch within the first few episodes.

4.3.5 SUMMARY

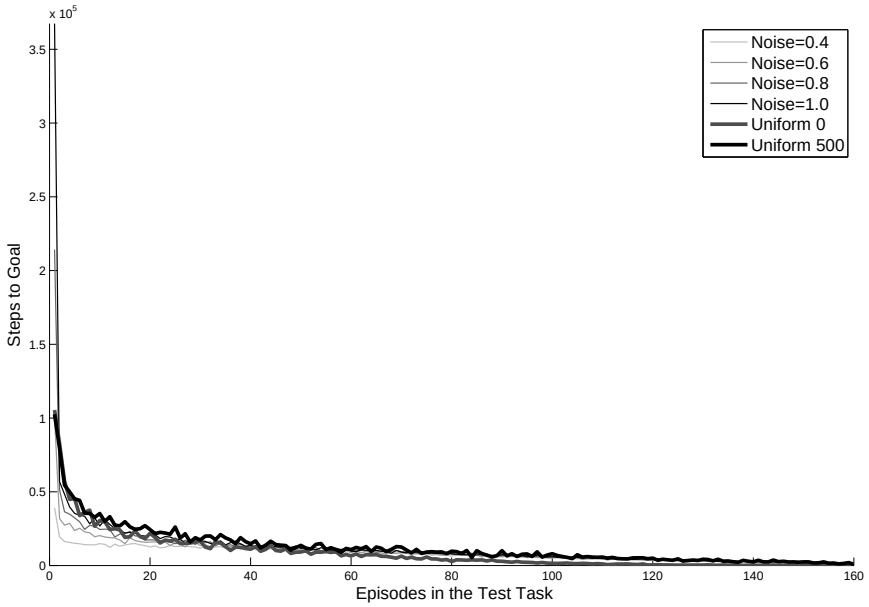
The first two experiments above show that an agent able to learn its own shaping rewards through training can use even a few training experiences to significantly improve its initial policy in a novel task. They also show that such training results in agents with convergence characteristics similar to that of agents using uniformly optimistic initial value functions. Thus, an agent that learns its own shaping rewards can improve its initial speed at solving a task when compared to an agent that cannot, but it will not converge to an approximately optimal policy in less time (as measured in episodes).

The results also seem to suggest that a better training environment is helpful but that its usefulness decreases as the signal predicting reward becomes more informative, and that increasing the complexity of the shaping function estimator does not appear to significantly improve the agent's performance. Although this is a very simple domain, this suggests that given a rich signal from which to predict reward, even a weak estimator of reward can greatly improve performance.

Finally, our third pair of experiments suggest that transfer is relatively robust to noise, both in the features themselves and in their relationship across tasks, resulting in performance benefits provided there is at least as much useful information in the features as there is noise. Beyond that, however, agents may experience negative transfer where either noisy features or an imperfect or approximate set of agent-space features result in poor learned shaping functions.



(a) The average number of steps required to first reach the test task goal given a predictor learned using features with imperfectly preserved semantics.



(b) Steps to reward against episodes in the test task for agents that have completed 20 training task episodes using features with imperfectly preserved semantics.

Figure 7: Results for the *noisy-semantics* task.

4.4 Keepaway

In this section we evaluate knowledge transfer using common features in Keepaway (Stone et al., 2005), a robot soccer domain implemented in the RoboCup soccer simulator. Keepaway is a challenging domain for reinforcement learning because it is multi-agent and has a high-dimensional continuous state space. We use Keepaway to illustrate the use of learned shaping rewards on a standard but challenging benchmark that has been used in other transfer studies (Taylor et al., 2007).

Keepaway has a square field of a given size, which contains players and a ball. Players are divided into two groups: keepers, who are originally in possession of the ball and try to stay in control of it, and takers, who attempt to capture the ball from the keepers. This arrangement is depicted in Figure 8.

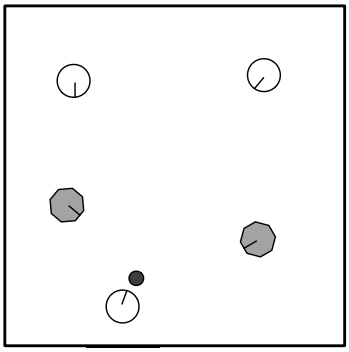


Figure 8: The Keepaway Task. The keepers (white circles) must keep possession of the ball and not allow the takers (gray octagons) to take it away. This diagram depicts 3v2 Keepaway, where there are 3 keepers and 2 takers.

Each episode begins with the takers in one corner of the field and the keepers randomly distributed. The episode ends when the ball goes out of bounds, or when a taker ends up in possession of the ball (i.e., within a small distance of the ball for a specified period of time). The goal of the keepers is then to maximize the duration of the episode. At each time step, the objective of learning is to modify the behavior of the keeper currently in possession of the ball. The takers and other keepers act according to simple hand-coded behaviors. Keepers not in possession of the ball try to open a clear shot from the keeper with the ball to themselves and attempt to receive the ball when it is passed to them. Takers either try to block keepers that are not holding the ball, try to take the ball from the keeper in possession, or try to intercept a pass.

Rather than using the primitive actions of the domain, keepers are given a set of predefined options. The options available to the keeper in possession of the ball are HoldBall (remain stationary while keeping the ball positioned away from the takers) and PassBall(k) (pass the ball to the kth other keeper). Since only the keeper in possession of the ball is acting according to the reinforcement learner at any given time, multiple keepers may learn during each episode; each keeper's learner runs separately.

The state variables are continuous and defined according to the center of the board and the location of the players, with the number of variables depending on the number of players. For example, 3v2 Keepaway (three keepers versus two takers) has thirteen state variables: the distance

from $K1$ (the keeper in possession) to each other player, the minimum angles BAC for each other keeper (where B is the other keeper, A is $K1$, and C is a taker—this measures how "open" each other keeper is to a pass), the distance from each player to the center, and the minimum distance from each other keeper to a taker. The number of state variables is $4K + 2T - 3$, for K keepers and T takers. We used a field measuring 20x20 units for 3v2 games, and a field measuring 30x30 for 4v3 and 5v4. For a more detailed description of the Keepaway domain we refer the reader to Stone et al. (2005).

4.4.1 EXPERIMENTAL STRUCTURE

In the previous section, we studied transferring portable shaping functions from a varying number of smaller randomly generated source tasks to a fixed larger target task. In Keepaway, instances of the domain are obtained by fixing the number of keepers and the number of takers. Since we cannot obtain experience in more than a few distinct source tasks, in this section we instead study the effect of varying amounts of training time in a source task on performance in a target task.

We thus studied transfer from 3v2 Keepaway to 4v3 and 5v4 Keepway, and from 4v3 to 5v4; these are the most common Keepaway configurations and are the same configurations studied by Taylor and Stone (2005). In all three cases we used the state variables from 5v4 as an agent-space. When a state variable is not defined (e.g., the distance to the 4th keeper in 3v2 Keepaway), we set distances and angles to keepers to 0, and distances and angles to takers to their maximum value, which effectively simulates their being present but not meaningfully involved in the current state. We employed linear function approximation with Sarsa (Sutton and Barto, 1998) using 32 radial basis functions per state variable, tiling each variable independently of the others, following and using the same parameters as Stone et al. (2005).

We performed 20 separate runs for each condition. We first ran 20 baseline runs for 3v2, 4v3, and 5v4 Keepaway, saving weights for the common space for each 3v2 and 4v3 run at 50, 250, 500, 1000, 2000, and 5000 episodes. Then for each set of common space weights from a given number of episodes, we ran 20 transfer runs. For example, for the 3v2 to 5v4 transfer with 250-episode weights, we ran 20 5v4 transfer runs, each of which used one of the 20 saved 250-episode 3v2 weights.

Because of Keepaway's high variance, and in order to provide results loosely comparable with Taylor and Stone (2005), we evaluated the performance of transfer in Keepaway by measuring the average time required to reach some benchmark performance. We selected a benchmark time for each setting (3v2, 4v3 or 5v4) which the baseline learner could consistently reach by about 5000 episodes. This benchmark time T is considered reached at episode n when the average of the times from $n-500$ to $n+500$ is at least T; this window averaging compensates Keepaway's high performance variance. The benchmark times for each domain were, in order, 12.5 seconds, 9.5 seconds, and 8.5 seconds.

4.4.2 RESULTS

Table 1 shows the results of performing transfer from 3v2 Keepaway to 4v3 Keepaway. Results are reported as time (in simulator hours) to reach the benchmark in the target task (4v3 Keepaway) given a particular number of training episodes in the source task (3v2 Keepaway), and the total time (source task training time plus time to reach the benchmark in the target task, in simulator hours). We can thereby evaluate whether the agents achieve *weak transfer*—where there is an improvement

in the target task with experience in the source task—by examining the third column (average 4v3 time), and *strong transfer*—where the sum of the time spent in both source and target tasks is lower than that taken when learning the target task in isolation—by examining the fourth column (average total time).

The results show that training in 3v2 decreases the amount of time required to reach the benchmark in 4v3, which shows that transfer is successful in this case and weak transfer is achieved. However, the total (source and target) time to benchmark never decreases with experience in the source task, so strong transfer is not achieved.

# 3v2 Episodes	Ave. 3v2 Time	Ave 4v3 Time	Ave. Total Time	Std. Dev.
0	0.0000	5.5616	**5.5616**	1.5012
50	0.0765	5.7780	5.8544	0.8870
250	0.3919	5.4763	5.8682	1.2399
500	0.8871	5.1192	6.0063	0.9914
1000	1.8166	4.7380	6.5546	1.2680
2000	3.9908	3.1295	7.1203	1.1465
5000	14.7554	**1.4236**	16.1790	0.2738

Table 1: Results of transfer from 3v2 Keepaway to 4v3 Keepaway.

Figure 9 shows sample learning curves for agents learning from scratch and agents using transferred knowledge from 5000 episodes of 3v2 Keepaway, demonstrating that agents that transfer knowledge start with better policies and learn faster.

Table 2 shows the results of transfer from 3v2 Keepaway (Table 1(a)) and 4v3 Keepaway (Table 1(b)) to 5v4 Keepaway. As before, in both cases more training on the easier task results in better performance in 5v4 Keepaway, demonstrating that weak transfer is achieved. However, the least total time (including training time on the source task) is obtained using a moderate amount of source task training, and so when transferring to 5v4 we achieve strong transfer.

Finally, Table 3 shows the results of transfer for shaping functions learned on both 3v2 and 4v3 Keepaway, applied to 5v4 Keepaway. Again, more training time obtains better results although over-training appears to be harmful.

These results show that knowledge transfer through agent-space can achieve effective transfer in a challenging problem and can do so in multiple problems through the same set of common features.

4.5 Discussion

The results presented above suggest that agents that employ reinforcement learning methods can be augmented to use their experience to learn their own shaping rewards. This could result in agents that are more flexible than those with pre-engineered shaping functions. It also creates the possibility of training such agents on easy tasks as a way of equipping them with knowledge that will make harder tasks tractable, and is thus an instance of an autonomous developmental learning system (Weng et al., 2000).

In some situations, the learning algorithm chosen to learn the shaping function, or the sensory patterns given to it, might result in an agent that is completely unable to learn anything useful. We do not expect such an agent to do much worse than one without any shaping rewards at all. Another

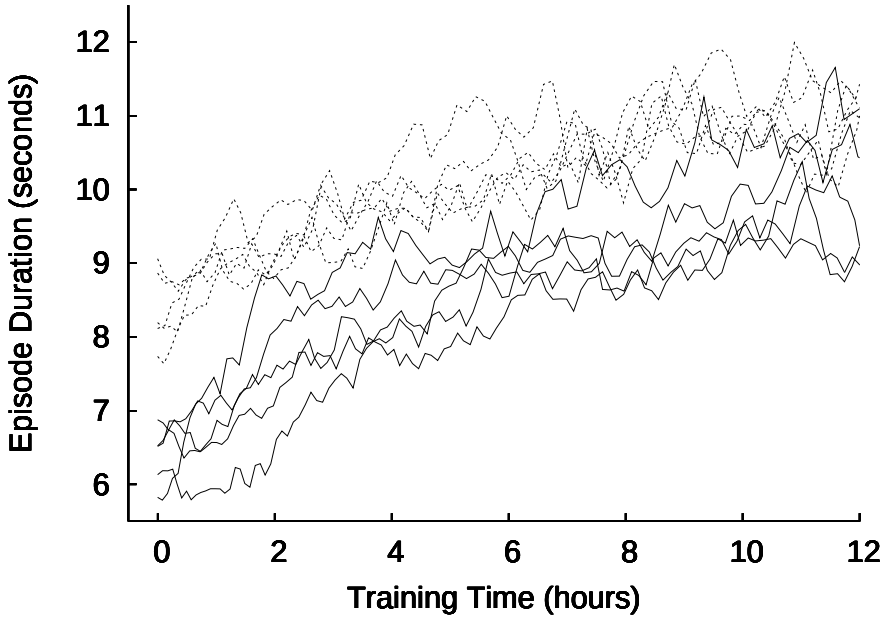


Figure 9: Sample learning curves for 4v3 Keepaway given no transfer (solid lines) or having experienced 5000 episodes of experience in 3v2 (dashed lines).

potential concern is the possibility that a maliciously chosen or unfortunate set of training tasks could result in an agent that performs worse than one with no training. Fortunately, such agents will still eventually be able to learn the correct value function (Ng et al., 1999).

All of the experiments reported in this paper use model-free learning algorithms. Given that an agent facing a sequence of tasks receives many example transitions between pairs of agent-space descriptors, it may prove efficient to instead learn an approximate transition model in agent-space and then use that model to obtain a shaping function via planning. However, learning a good transition model in such a scenario may prove difficult because the agent-space features are not Markov.

In standard classical search algorithms such as A*, a heuristic imposes an order in which nodes are considered during the search process. In reinforcement learning the state space is searched by the agent itself, but its initial value function (either directly or via a shaping function) acts to order the selection of unvisited nodes by the agent. Therefore, we argue that reinforcement learning agents using non-uniform initial value functions are using something very similar to a heuristic, and those that are able to learn their own portable shaping functions are in effect able to learn their own heuristics.

5. Skill Transfer

The previous section showed that we can effectively transfer knowledge about reward when a sequence of tasks is related and reward-linked, and that such knowledge can significantly improve

(a) Transfer results from 3v2 to 5v4 Keepaway.

# 3v2 Episodes	Ave. 3v2 Time	Ave 5v4 Time	Ave. Total Time	Std. Dev.
0	0.0000	7.4931	7.4931	1.5229
50	0.0765	6.3963	6.4728	1.0036
250	0.3919	5.6675	6.0594	0.7657
500	0.8870	5.9012	6.7882	1.1754
1000	1.8166	3.9817	**5.7983**	1.2522
2000	3.9908	3.9678	7.9586	1.8367
5000	14.7554	**3.9241**	18.6795	1.3228

(b) Transfer results from 4v3 to 5v4 Keepaway.

# 4v3 Episodes	Ave. 4v3 Time	Ave 5v4 Time	Ave. Total Time	Std. Dev.
0	0.0000	7.4931	7.4930	1.5229
50	0.0856	6.6268	6.7125	1.2162
250	0.4366	6.1323	**6.5689**	1.1198
500	0.8951	6.3227	7.2177	1.0084
1000	1.8671	6.0406	7.9077	1.0766
2000	4.0224	5.0520	9.0744	0.9760
5000	11.9047	**3.218**	15.1222	0.6966

Table 2: Results of transfer to 5v4 Keepaway.

# 3v2 Episodes	# 4v3 Episodes	Ave 5v4 Time	Ave. Total Time	Std. Dev.
500	500	6.1716	8.0703	1.1421
500	1000	5.6139	8.6229	0.9597
1000	500	**4.5395**	**7.3922**	0.6689
1000	1000	4.8648	8.8448	0.9517

Table 3: Results of transfer from both 3v2 Keepaway and 4v3 Keepaway to 5v4 Keepaway.

performance. We can apply the same framework to effect skill transfer by creating portable option policies. Most option learning methods work within the same state space as the problem the agent is solving at the time. Although this can lead to faster learning on later tasks in the same state space, learned options would be more useful if they could be reused in later tasks that are related but have distinct state spaces.

In this section we demonstrate empirically that an agent that learns portable options directly in agent-space can reuse those options in future related tasks to significantly improve performance. We also show that the best performance is obtained using portable options in conjunction with problem-specific options.

5.1 Options in Agent-Space

Following section 4.2, we consider an agent solving n problems $M_1, \ldots, M_n$ with state spaces $S_1, \ldots, S_n$, and action space A. Once again, we associate a four-tuple σ_i^j with the ith state in M_j:

$$\sigma_i^j = \langle s_i^j, d_i^j, r_i^j, v_i^j \rangle,$$

where s_i^j is the usual problem-space state descriptor (sufficient to distinguish this state from the others in S_j), d_i^j is the agent-space descriptor, r_i^j is the reward obtained at the state and v_i^j is the state's value (expected total reward for action starting from the state).

The agent is also either given, or learns, a set of higher-level options to reduce the time required to solve the task. Options defined using s_i^j are not portable between tasks because the form and meaning of s_i^j (as a problem-space descriptor) may change from one task to another. However, the form and meaning of d_i^j (as an agent-space descriptor) does not. Therefore we define agent-space option components as:

$$\begin{array}{lll} \pi_o: & (d_i^j, a) & \mapsto [0,1], \\ I_o: & d_i^j & \mapsto \{0,1\}, \\ \beta_o: & d_i^j & \mapsto [0,1]. \end{array}$$

Although the agent will be learning task and option policies in different spaces, both types of policies can be updated simultaneously as the agent receives both agent-space and problem-space descriptors at each state.

To support learning a portable shaping function, an agent space should contain some features that are correlated to return across tasks. To support successful skill policy learning, an agent space needs more: it must be suitable for directly learning control policies.

If that is the case, then why not perform task learning (in addition to option learning) in agent-space? There are two primary reasons why we might prefer to perform task learning in problem-space, even when given an agent-space suitable for control learning. The first is that agent-space may be very much larger than problem-space, making directly learning the entire task in agent-space inefficient or impractical. The second is that the agent-space may only be sufficient for learning control policies locally, rather than globally. In the next two sections we demonstrate portable skill learning on domains with each characteristic in turn.

5.2 The Lightworld Domain

The lightworld domain is a parameterizable class of discrete domains which share an agent-space that is much larger than any individual problem-space. In this section, we empirically examine whether learning portable skills can improve performance in such a domain.

An agent is placed in an environment consisting of a sequence of rooms, with each room containing a locked door, a lock, and possibly a key. In order to leave a room, the agent must unlock the door and step through it. In order to unlock the door, it must move up to the lock and press it, but if a key is present in the room the agent must be holding it to successfully unlock the door. The agent can obtain a key by moving on top of it and picking it up. The agent receives a reward of 1000 for leaving the door of the final room, and a step penalty of -1 for each action. Six actions are available: movement in each of the four grid directions, a pickup action and a press action. The environments are deterministic and unsuccessful actions (for example, moving into a wall) result in no change in state.

In order to specify an individual lightworld instance, we must specify the number of rooms, x and y sizes for each room, and the location of the room entrance, key (or lack therefore), lock and door in each. Thus, we may generate new lightworld instances by generating random values for each of these parameters.

We equip the agent with twelve light sensors, grouped into threes on each of its sides. The first sensor in each triplet detects red light, the second green and the third blue. Each sensor responds to light sources on its side of the agent, ranging from a reading of 1 when it is on top of the light source, to 0 when it is 20 squares away. Open doors emit a red light, keys on the floor (but not those held by the agent) emit a green light, and locks emit a blue light. Figure 10 shows an example.

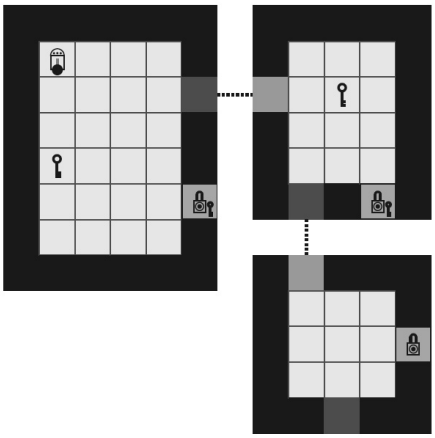


Figure 10: A small example lightworld.

Five pieces of data form a problem-space descriptor for any lightworld instance: the current room number, the x and y coordinates of the agent in that room, whether or not the agent has the key, and whether or not the door is open. We use the light sensor readings as an agent-space because their semantics are consistent across lightworld instances. In this case the agent-space (with 12 continuous variables) has much higher dimension than any individual problem-space, and it is impractical to perform task learning in it directly, even though the problem might in principle be solvable that way.

5.2.1 TYPES OF AGENT

We used five types of reinforcement learning agents: agents without options, agents with problem-space options, agents with perfect problem-space options, agents with agent-space options, and agents with both option types.

The agents without options used Sarsa(λ) with ε-greedy action selection ($\alpha = 0.1$, $\gamma = 0.99$, $\lambda = 0.9$, $\varepsilon = 0.01$) to learn a solution policy in problem-space, with each state-action pair assigned an initial value of 500.

Agents with problem-space options had an (initially unlearned) option for each pre-specified salient event (picking up each key, unlocking each lock, and walking through each door). Options were learned in problem-space and used the same parameters as the agent without options, but used

off-policy trace-based tree-backup updates (Precup et al., 2000) for intra-option learning. We used an option termination reward of 1 for successful completion, and a discount factor of 0.99 per action. Options could be executed only in the room in which they were defined and only in states where their value function exceeded a minimum threshold (0.0001). Because these options were learned in problem-space, they were useful but needed to be relearned for each individual lightworld.

Agents with perfect problem-space options were given options with pre-learned policies for each salient event, though they still performed option updates and were otherwise identical to the standard agent with options. They represent the ideal case of agents with that can perform perfect transfer, arriving in a new task with fully learned options.

Agents with agent-space options still learned their solution policies in problem-space but learned their option policies in agent-space. Each agent employed three options: one for picking up a key, one for going through an open door and one for unlocking a door, with each one's policy a function of the twelve light sensors. Since the sensor outputs are continuous we employed linear function approximation for each option's value function, performing updates using gradient descent ($\alpha = 0.01$) and off-policy trace-based tree-backup updates. We used an option termination reward of 1, a step penalty of 0.05 and a discount factor of 0.99. An option could be taken at a particular state when its value function there exceeded a minimum threshold of 0.1. Because these options were learned in agent-space, they could be transferred between lightworld instances.

Finally, agents with both types of options were included to represent agents that learn both general portable and specific non-portable skills simultaneously.

Note that all agents used discrete problem-space value functions to solve the underlying task instance, because their agent-space descriptors are only Markov in lightworlds with a single room, which were not present in our experiments.

5.2.2 EXPERIMENTAL STRUCTURE

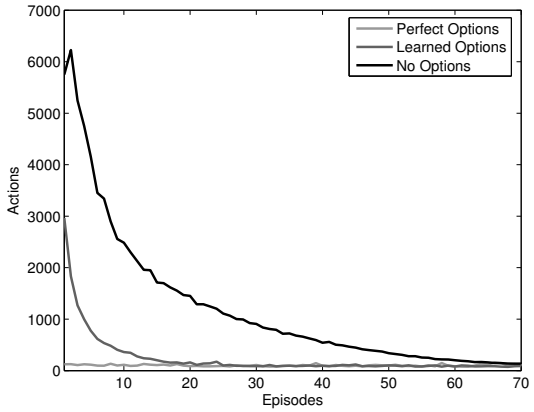
We generated 100 random lightworlds, each consisting of 2-5 rooms with width and height of between 5 and 15 cells. A door and lock were randomly placed on each room boundary, and $\frac{1}{3}$ of rooms included a randomly placed key. This resulted in state space with between 600 and approximately $20,000$ state-action pairs ($4,900$ on average). We evaluated each problem-space option agent type on 1000 lightworlds (10 samples of each generated lightworld).

To evaluate the performance of agent-space options as the agents gained more experience, we similarly obtained 1000 lightworld samples and test tasks, but for each test task we ran the agents once without training and then with between 1 and 10 training experiences. Each training experience for a test lightworld task consisted of 100 episodes in a training lightworld randomly selected from the remaining 99. Although the agents updated their options during evaluation in the test lightworld, these updates were discarded before the next training experience so the agent-space options never received prior training in the test lightworld.

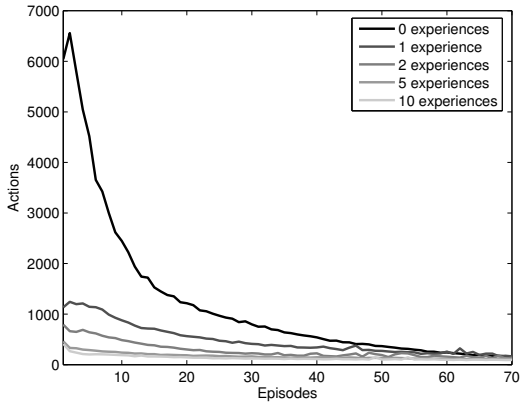
5.2.3 RESULTS

Figure 11(a) shows average learning curves for agents employing problem-space options, and Figure 11(b) shows the same for agents employing agent-space options. The first time an agent-space option agent encounters a lightworld, it performs similarly to an agent without options (as evidenced by the two topmost learning curves in each figure), but its performance rapidly improves with experience in other lightworlds. After experiencing a single training lightworld, the agent starts with

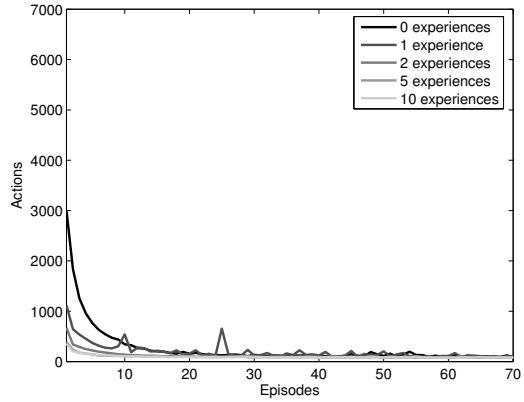
better performance than an agent using problem-space options alone, until by 5 experiences its learning curve is similar to that of an agent with perfect problem-space options (compare the learning curves in Figure 11(b) with the bottom-most learning curve of Figure 11(a)), even though its options are never trained in the same lightworld in which it is tested. The comparison between Figures 11(a) and 11(b) shows that agent-space options can be successfully transferred between lightworld instances.



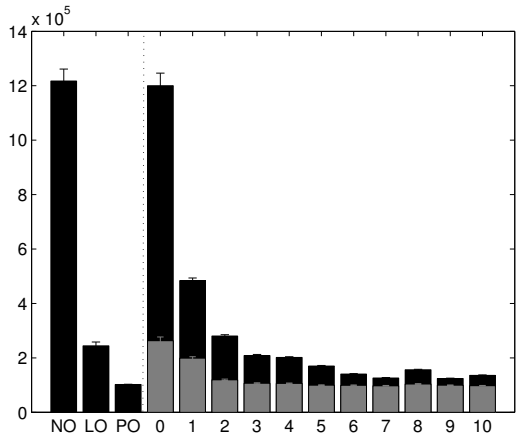
(a) Learning curves for agents with problem-space options.



(b) Learning curves for agents with agent-space options, with varying numbers of training experiences.



(c) Learning curves for agents with agent-space *and* problem-space options, with varying numbers of training experiences.



(d) Total steps over 70 episodes for agents with no options (NO), learned problem-space options (LO), perfect options (PO), agent-space options with 0-10 training experiences (dark bars), and both option types with 0-10 training experiences (light bars).

Figure 11: Results for the Lightworld Domain.

Figure 11(c) shows average learning curves for agents employing *both* types of options.[4] The first time such agents encounter a lightworld, they perform as well as agents using problem-space

4. In 8 of the more than 200,000 episodes used when testing agents with both types of options, an agent-space value function approximator diverged, and we restarted the episode. Although this is a known problem with the backup

options (compare with the second highest curve in Figure 11(a)), and thereafter rapidly improve their performance (performing better than agents using only agent-space options) and again by 5 experiences performed nearly as well as agents with perfect options. This improvement can be explained by two factors. First, the agent-space is much larger than any individual problem-space, so problem-space options are easier to learn from scratch than agent-space options. This explains why agents using only agent-space options and no training experiences perform more like agents without options than like agents with problem-space options. Second, options learned in our problem-space can represent exact solutions to specific subgoals, whereas options learned in our agent-space are general and must be approximated, and are therefore likely to be slightly less efficient for any specific subgoal. This explains why agents using both types of options perform better in the long run than agents using only agent-space options.

Figure 11(d) shows the mean total number of steps required over 70 episodes for agents using no options, problem-space options, perfect options, agent-space options, and both option types. Experience in training environments rapidly drops the number of total steps required to nearly as low as the number required for an agent with perfect options. It also clearly shows that agents using both types of options do consistently better than those using agent-space options alone. We note that the error bars in Figure 11(d) are small and decrease with experience, indicating consistent transfer.

In summary, these results show that learning using portable options can greatly improve performance over learning using problem-specific options. Given enough experience, learned portable options can perform similarly to perfect pre-learned problem-specific options, even when the agent-space is much harder to learn in than any individual problem-space. However, the best learning strategy is to learn using both problem-specific options and portable options.

5.3 The Conveyor Belt Domain

In the previous section we showed that an agent can use experience in related tasks to learn portable options, and that those options can improve performance in later tasks, when the agent has a high-dimensional agent-space. In this section we consider a task where the agent-space is not high-dimensional, but is only sufficient for local control.

In the conveyor belt domain, a conveyor belt system must move a set of objects from a row of feeders to a row of bins. There are two types of objects (triangles and squares), and each bin starts with a capacity for each type. The objects are issued one at a time from a feeder and must be directed to a bin. Dropping an object into a bin with a positive capacity for its type decrements that capacity.

Each feeder is directly connected to its opposing bin through a conveyor belt, which is connected to the belts above and below it at a pair of fixed points along its length. The system may either run the conveyor belt (which moves the current object one step along the belt) or try to move it up or down (which only moves the object if it is at a connection point). Each action results in a reward of -1, except where it causes an object to be dropped into a bin with spare capacity, in which case it results in a reward of 100. Dropping an object into a bin with zero capacity for that type results in the standard reward of -1.

method we used (Precup et al., 2000), it did not occur during the same number of samples obtained for agents with agent-space options only.

To specify an instance of the conveyor belt domain, we must specify the number of objects present, belts present, bin capacities, belt length, and where each adjacent pair of belts are connected. A small example conveyor belt system is shown in Figure 12.

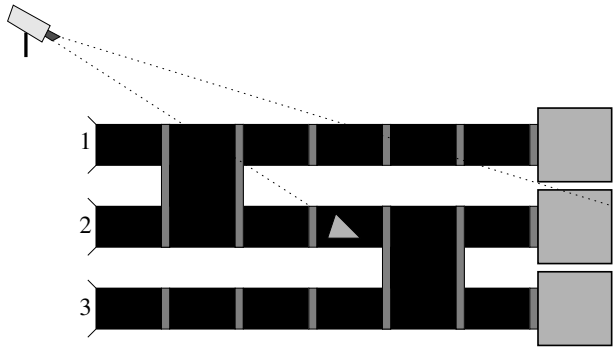


Figure 12: A small example conveyor belt problem.

Each system has a camera that tracks the current object and returns values indicating the distance (up to 15 units) to the bin and each connector along the current belt. Because the space generated by the camera is present in every conveyor-belt problem and retains the same semantics, it is an agent-space, and because it is discrete and relatively small ($13,500$ states), we can learn policies in it without function approximation. However, because it is non-Markov (due to its limited range and inability to distinguish between belts), it cannot be used as a problem-space.

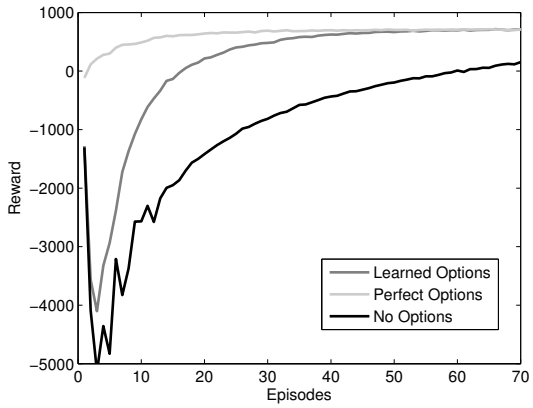
A problem-space descriptor for a conveyor belt instance consists of three numbers: the current object number, the belt it is on, and how far along that belt it lies (technically we should include the current capacity of each bin, but we can omit this and still obtain good policies). We generated 100 random instances with 30 objects and 20-30 belts (each of length 30-50) with randomly-selected interconnections, resulting in problem-spaces of $18,000$-$45,000$ states.

We ran experiments where the agents learned three options: one to move the current object to the bin at the end of the belt it is currently on, one for moving it to the belt above it, and one for moving it to the belt below it. We used the same agent types and experimental structure as before, except that the agent-space options did not use function approximation.

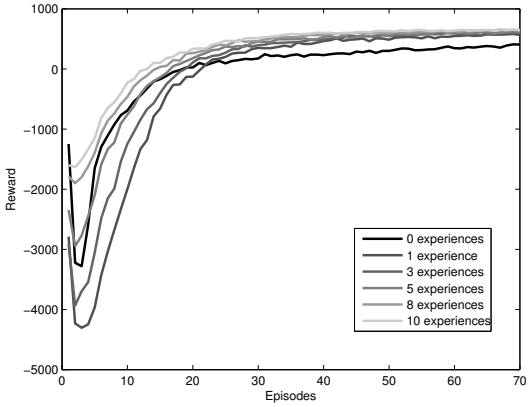
5.3.1 RESULTS

Figures 13(a), 13(b) and 13(c) show learning curves for agents employing no options, problem-space options and perfect options; agents employing agent-space options; and agents employing both types of options, respectively.

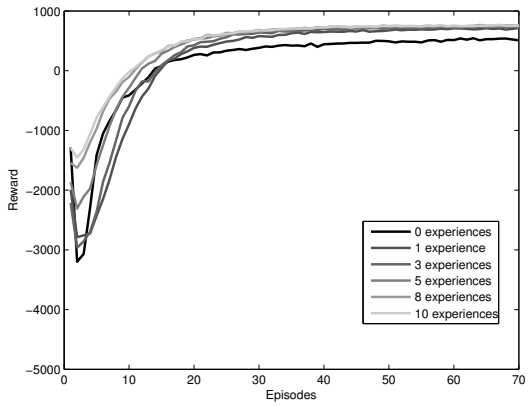
Figure 13(b) shows that the agents with agent-space options and no prior experience initially improve quickly but eventually obtain lower quality solutions than agents with problem-space options (Figure 13(a)). One or two training experiences result in roughly the same curve as agents using problem-space options, but by 5 training experiences the agent-space options are a significant improvement (although due to their limited range they are never as good as perfect options). This initial dip relatively to agents with no prior experience is probably due to the limited range of the agent-space options (due to the limited range of the camera) and the fact that they are only locally Markov, even for their own subgoals.



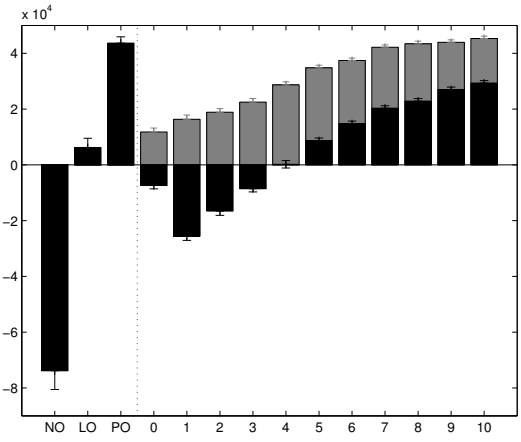
(a) Learning curves for agents with problem-space options.



(b) Learning curves for agents with agent-space options, with varying numbers of training experiences.



(c) Learning curves for agents with both types of options, with varying numbers of training experiences.



(d) Total reward over 70 episodes for agents with no options (NO), learned problem-space options (LO), perfect options (PO), agent-space options with 0-10 training experiences (dark bars), and both option types with 0-10 training experiences (light bars).

Figure 13: Results for the Conveyor Belt Domain.

Figure 13(c) shows that agents with both option types do not experience this initial dip relative to agents with no prior experience and outperform problem-space options immediately, most likely because the agent-space options are able to generalise across belts. Figure 13(d) shows the mean total reward for each type of agent. Agents using agent-space options eventually outperform agents using problem-space options only, even though the agent-space options have a much more limited range; agents using both types of options consistently outperform agents with either option type and eventually approach the performance of agents using pre-learned problem-space options.

In summary, these results demonstrate that when an agent-space is only locally Markov, learning portable options can still result in a significant performance improvement over learning using problem-specific options, but that even with a great deal of experience will not reach the perfor-

mance of perfect pre-learned problem-specific options. Once again, the best approach is to learn using both problem-specific and agent-space options simultaneously.

5.4 Summary

Our results show that options learned in agent-space can be successfully transferred between related tasks, and that this significantly improves performance in sequences of tasks where the agent space cannot be used for learning directly. Our results suggest that when the agent space is large but can support global policies, experience in related tasks can eventually result in options that perform as well as perfect problem-specific options. When the agent space is only locally Markov, learned portable options will improve performance but are unlikely to reach the performance of perfect problem-specific options due to their limited range.

We expect that, in general, learning an option in agent-space will often actually be harder than solving an individual problem-space instance, as was the case in our experiments. In such situations, learning both problem-specific and agent space options simultaneously will likely obtain better performance than either individually. Since intra-option learning methods allow for the update of several options from the same experiences, it may be better in general to simultaneously learn both general portable skills and specific, exact but non-portable skills, and allow them to bootstrap each other.

6. Related Work

Although the majority of research in transfer assumes that the source and target problems have the same state space, some existing research does not make that assumption.

Wilson et al. (2007) consider the case where an agent faces a sequence of environments, each generated by one of a set of environment classes. Each environment class is modeled as a distribution of values of some observed signal given a feature vector, and since the number of classes is unknown, the agent must learn an infinite mixture model of classes. When faced with a new environment, the agent determines which of its existing models it best matches or whether it instead corresponds to a novel class. A model-based planning algorithm is then used to solve the new task. This work explicitly considers environment sequences that do not have the same state space, and thus defines the distributions of each environment class over the output of a function f that generates a feature vector for each state in each environment. Since that feature vector retains its semantics across all of the tasks, it is exactly an agent-space descriptor as defined here. Thus, this work can be seen as using agent-space to learn a set of environment models.

Banerjee and Stone (2007) consider transfer learning for the case of General Game Playing, where knowledge gained from playing one game (e.g., Tic-Tac-Toe) is exploited to improve performance in another (e.g., Connect-4). Here, transfer is affected through the game tree: the authors define generic game-tree features that apply across all games and then use their Q-values to initialize the values of novel states with matching features when playing a subsequent game. This is a very similar mechanism to a portable shaping function, including the use of features—in this case derived from the game tree—that are common across all tasks.

Taylor et al. (2007) use a hand-coded transfer function to seed one task's value function with learned values from another similar task with a potentially different state space. This requires a mapping to be constructed between the weights of the function approximators of each pair of tasks

between which transfer might occur.[5] Our method offers two advantages over this. First, we effectively require the construction of a mapping from each task to an agent-space, so the number of mappings scales linearly with the number of tasks, rather than quadratically. Second, through the use of a shaping function, those mappings can be constructed between state descriptors, rather than between function approximation terms. This allows us to treat the function approximator employed for each task as a black box rather than requiring detailed knowledge of its construction, and it allows us to transfer between potentially very different function approximators where a direct mapping might be difficult to obtain. On the other hand, if performance is critical, then constructing a specialized task-to-task mapping may result in better performance than a more generic agent-space mapping; the results in Taylor et al. (2007) seem slightly better than those given in Section 4.4.2, although a direct comparison is not possible since the benchmarks used (expressing the underlying learning performance) differ (presumably due to implementation differences), even though we used the same parameters.

Another related line of research focuses on effecting representation transfer, where basis functions are learned in one task and applied in another. Representation transfer has so far focused primarily on task sequences where reward function or dynamics differ but the state space remains the same (Ferguson and Mahadevan, 2006; Ferrante et al., 2008). If the state spaces differ significantly, manifold alignment or scaling methods may be employed to transform basis functions from one state space to another (Ferguson and Mahadevan, 2008); however, such transformations require prior knowledge of the topology of the two state spaces to either achieve scaling or to obtain a good alignment.

Lazaric et al. (2008) introduced sample transfer, where sample transitions from a source task may be used as additional data to improve performance in a new task. Transition samples from a set of source tasks are stored, and then used along with a small set of sample transitions in a new task to compute a similarity measure between the new task and the source tasks. The transferred transitions are then sampled according to the similarity measure and added to the new task samples, resulting in a performance boost for batch-learning methods. Reusing such samples requires their state descriptors to (at least) be the same size, although if the reused descriptors were defined in an agent-space, then such a method may be useful for more efficiently learning portable shaping functions.

Konidaris and Hayes (2004) describe a similar method to ours that uses training tasks to learn associations between reward and strong signals at reward states, resulting in a significant improvement in the total reward obtained by a simulated robot learning to find a puck in a novel maze. The research presented in this paper employs a more general mechanism where the agent learns a heuristic from all visited states.

Zhang and Dietterich (1995) use common features to transfer learned value functions across a class of job-shop scheduling problems. The value functions (represented as neural networks) were learned using TD(λ) over a set of features constructed to be common to the entire problem class. Value functions trained using small instances of scheduling problems were then used to obtain solutions to larger problems. This research is a case where an agent-space was sufficient to

5. Construction has been primarily accomplished by hand, but we briefly discuss recent work aimed at automating it in Section 7.1.

represent a solution to each individual problem and the need for a problem-specific state space was avoided.[6]

The X-STAGE algorithm (Boyan and Moore, 2000) uses features common across a class of tasks to transfer learned evaluation functions that predict the performance of a local search algorithm applied to an optimization task. The evaluation functions—which are similar to value functions in that they predict the outcome of the execution of a policy, in this case a search algorithm—serve to identify the most promising restart points for local search. The X-STAGE algorithm learns a distinct evaluation function for each source task and then obtains a "vote" for the next action in the target task from each source evaluation function. Interestingly, while this method of transfer results in an initial performance boost, it eventually obtains solutions inferior to those obtained by learning a problem-specific evaluation function; our use of shaping avoids this dilemma, because it naturally incorporates experience from the current task into the agent's value function and thus avoids permanent bias arising from the use of transferred knowledge.

All of the option creation methods given in Section 2.2 learn options in the same state space in which the agent is performing reinforcement learning, and thus the options can only be reused for the same problem or for a new problem in the same space. The available state abstraction methods (Jonsson and Barto, 2001; Hengst, 2002) only allow for the automatic selection of a subset of this space for option learning, or they require an explicit transformation from one space to another (Ravindran and Barto, 2003a).

There has been some research focusing on extracting options by exploiting commonalities in collections of policies (Thrun and Schwartz, 1995; Bernstein, 1999; Perkins and Precup, 1999; Pickett and Barto, 2002) or analysing the relationships between variables given sample trajectories (Mehta et al., 2008), but in each case the options are learned over a single state space. In contrast, we leave the method used for creating the options unspecified—any option creation method may be used—but create them in a portable space.

Fernández and Veloso (2006) describe a method called Policy Reuse, where an agent given a library of existing policies determines which of them is closest to a new problem it faces, and then incorporates that policy into the agent's exploration strategy. The resulting distance metric is also used to build a library of core policies that can be reused for later tasks in the same state space. Although this method has very attractive attributes (particularly when applied in a hierarchical setting), it is limited to task sequences where only the reward function changes.

Torrey et al. (2006) show that policy fragments learned in a symbolic form using inductive logic programming (ILP) can be transferred to new tasks as constraints on the new value-function. This results in a substantial performance improvement. However, a user must provide a mapping from state variables in the first task to the second, and the use of an ILP implementation introduces significant complexity and overhead.

Croonenborghs et al. (2007) learns relational options and shows that they can be transferred to different state spaces provided the same symbols are still present. This approach is similar to ours, in that we could consider the symbols shared between the tasks to be an agent-space.

6. The agent-space in this case did introduce aliasing, which occasionally caused policies with loops. This was avoided using a loop-detection algorithm.

7. Discussion

The work in the preceding sections has shown that both knowledge and skill transfer can be effected across a sequence of tasks through the use of features common to all tasks in the sequence. Our results have shown significant improvements over learning from scratch, and the framework offers some insight into which problems are amenable to transfer.

However, our framework requires the identification of a suitable agent-space to facilitate transfer, but it does not specify how that space is identified, which creates a design problem similar to that of standard state space design. Researchers in the reinforcement learning community have so far developed significant expertise at designing problem-spaces, but not agent-spaces. Nevertheless, the example domains in this paper offer several examples of related tasks with different types of common feature sets—deictic sensors (the Rod positioning task and the Lightworld), a maximum set (Keepaway), and local sensing (the Conveyor Belt domain)—and we have pointed out the use of similar feature sets in existing work (Zhang and Dietterich, 1995; Boyan and Moore, 2000; Wilson et al., 2007; Snel and Whiteson, 2010). Taken together, these examples suggest that transfer via common features may find wide application.

Additionally, for option learning, an agent-space descriptor should ideally be Markov within the set of states that the option is defined over. The agent-space descriptor form will therefore affect both what options can be learned and their range. In this respect, designing agent-spaces for learning options requires more care than for learning shaping functions.

An important assumption made in our option transfer work is that all tasks have the same set of available actions, even though they have different state spaces. If this is not the case, then learning portable options directly is only possible if the action spaces share a common subset or if we can find a mapping between action spaces. If no such mapping is given, we may be able to construct one from experience using a homomorphism (Ravindran and Barto, 2003b).

When learning portable shaping functions, if the action space differs across tasks then we can simply learn shaping functions defined over states only (as potential-based shaping functions were originally defined by Ng et al., 1999) rather than defining them over state-action pairs. Although we expect that learning using portable state-only shaping functions will not perform as well as learning using portable state-action shaping functions, we nevertheless expect that they will result in substantial performance gains for reward-linked tasks.

The idea of an agent-centric representation is closely related to the notion of deictic or egocentric representations (Agre and Chapman, 1987), where objects are represented from the point of view of the agent rather than in some global frame of reference. We expect that for most problems, especially in robotics, agent-space representations will be egocentric, except in manipulation tasks, where they will likely be object-centric. In problems involving spatial maps, we expect that the difference between problem-space and agent-space will be closely related to the difference between allocentric and egocentric representations of space (Guazzelli et al., 1998)—the utility of such spaces for transfer has been demonstrated by Frommberger (2008).

7.1 Identifying Agent-Spaces

In this work we have assumed that an agent-space is given. However, this may not always be the case; if it is not, then we are faced with the problem of attempting to automatically identify collections of features that retain their semantics across tasks.

This problem may arise in several settings. In the simplest setting, given a pair of corresponding feature sets for two problems, we must determine whether the two feature sets are an agent-space. To do this, we may build approximate transition models for each feature set, and then compare them.

In an intermediate setting, we might be given two sets of corresponding features and asked to identify which subsets of these features form an agent-space. Snel and Whiteson (2010) report very promising results on this problem using a formalization of how task-informative a feature is (and thus how likely it is to be in problem-space) against how domain-informative it is (and thus how likely it is to be in agent-space).

The problem becomes much harder when we are given an arbitrary number of features for each task, and we are required to both identify correspondences between features and determine which subset of features form an agent-space. Taylor et al. (2008) address a similar problem: constructing mappings between two sets of state variables for a pair of given tasks. They propose an algorithm which generates all possible mappings from the first task to the second, then learns a transition model from the first and compares its predictions (using each candidate mapping) to sample data from the second; finally the algorithm selects the mapping with the lowest transition error. This method can be adapted to our setting by selecting a reference task (most likely the first task the agent sees) and then building mappings from each new task back to it. The subset of variables in the reference task that appear in all mappings constitute an agent-space.

Taylor et al. (2008) claim that their algorithm is data-efficient because the same sample transitions can be used for comparing every possible mapping, even though the algorithm's time complexity is exponential in the size of the number of variables in the two tasks. In our setting, once the first mapping (from the reference task to some other task) has been constructed, we may remove the reference variables absent from the mapping from later consideration, which could lead to significant efficiency gains when constructing later mappings. In addition, such a method (mapping to a reference task) would require only $n-1$ mappings to be constructed for arbitrary transfer between pairs of tasks drawn from a sequence of n tasks, whereas a direct mapping methodology requires $O(n^2)$ mappings to be constructed.

In the most difficult setting, we might be given no features at all, and asked to construct an agent-space. This can be considered a problem of discovering latent variables that describe aspects of the state space which can be used for transfer. We expect that this will be a challenging but fruitful avenue of future work.

7.2 Identifying Reward-Linked Tasks

An important distinction raised by this work is the difference between related tasks and tasks that are both related and reward-linked. Tasks that are related (in that they share an agent-space) but are not reward-linked do not allow us to transfer knowledge about the value function, since they do not necessarily have similar reward functions.

This raises an important question for future work: given a solved task T_s and a new related task T_n, how can we determine whether they are reward-linked? More broadly, given a set of previously learned related tasks that are not reward-linked, which one should we use as the source of a portable shaping function for a new related task?

Answering these questions relies upon us finding some method of comparison for the two task reward functions, R_n and R_s. Since one of the important motivations behind learning portable shaping functions is boosting initial task performance, we would prefer to perform this comparison

without requiring much experience. If we are given R_n in some appropriate functional form, then we could obtain its value at sample state-action pairs in T_s and compare the results with the experienced values of R_s. If the method used to compare the two reward functions returns a distance metric, then the agent could use it to cluster portable shaping functions and build libraries of them, drawing on an appropriate one for each new task it encounters.

However, if we are not given R_n, then we must sample R_n with experience. It is easy to construct an adversarial argument showing that an agent with experience only in T_s cannot determine whether T_n and T_s are reward-linked without at the very least one full episode's worth of experience in T_n.

However, we do not believe the complete absence of prior information about a task is representative of applied reinforcement learning settings where the agent must solve multiple tasks sequentially. In most cases, the agent has access to some extra information about a new task before it attempts to solve it. We can formalize this by attaching a descriptor to each task; then the extent to which an agent can recognize a task "type" depends on how much information is contained in its descriptor. If the relationship between task descriptor and task "type" is not known in advance, it can be learned over time using training examples obtained by comparing reward functions after experience.

8. Summary and Conclusions

We have presented a framework for transfer in reinforcement learning based on the idea that related tasks share common features and that transfer can take place through functions defined over those related features. The framework attempts to capture the notion of tasks that are related but distinct, and it provides some insight into when transfer can be usefully applied to a problem sequence and when it cannot.

Most prior work on transfer relies on mappings between pairs of tasks, and therefore implicitly defines transfer as a relationship between problems. This work provides a contrasting viewpoint by relying on a stronger notion of an agent: that there is something common across a series of tasks faced by the same agent, and that that commonality derives from features shared because they are a property of an agent.

We have empirically demonstrated that this framework can be successfully applied to significantly improve transfer using both knowledge transfer and skill transfer. It provides a practical approach to building agents that are capable of improving their own problem-solving capabilities through experience over multiple problems.

Acknowledgments

We would like to thank Ron Parr and our three anonymous reviewers for their tireless efforts to improve this work. We also thank Gillian Hayes, Colin Barringer, Sarah Osentoski, Özgür Şimşek, Michael Littman, Aron Culotta, Ashvin Shah, Chris Vigorito, Kim Ferguson, Andrew Stout, Khashayar Rohanimanesh, Pippin Wolfe and Gene Novark for their comments and assistance. Andrew Barto and George Konidaris were supported in part by the National Science Foundation under Grant No. CCF-0432143, and Andrew Barto was supported in part by a subcontract from Rutgers University, Computer Science Department, under award number HR0011-04-1-0050 from DARPA. George Konidaris was supported in part by the AFOSR under grant AOARD-104135

and the Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, DARPA, the AFOSR, or the Singapore Ministry of Education.

References

P.E. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, 1987.

B. Banerjee and P. Stone. General game learning using knowledge transfer. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 672–677, 2007.

A.G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13:41–77, 2003. Special Issue on Reinforcement Learning.

D.S. Bernstein. Reusing old policies to accelerate learning on new MDPs. Technical Report UM-CS-1999-026, Department of Computer Science, University of Massachusetts at Amherst, April 1999.

J. Boyan and A.W. Moore. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:77–112, 2000.

T. Croonenborghs, K. Driessens, and M. Bruynooghe. Learning relational options for inductive transfer in relational reinforcement learning. In *Proceedings of the Seventeenth International Conference on Inductive Logic Programming*, pages 88–97, 2007.

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

B.L. Digney. Learning hierarchical control structures for multiple tasks and changing environments. In R. Pfeifer, B. Blumberg, J. Meyer, and S.W. Wilson, editors, *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Zurich, Switzerland, August 1998. MIT Press.

M. Dorigo and M. Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press/Bradford Books, 1998.

K. Ferguson and S. Mahadevan. Proto-transfer learning in Markov Decision Processes using spectral methods. In *Proceedings of the ICML Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh, June 2006.

K. Ferguson and S. Mahadevan. Proto-transfer learning in Markov Decision Processes using spectral methods. Technical Report TR-08-23, University of Massachusetts Amherst, 2008.

F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, 2006.

E. Ferrante, A. Lazaric, and M. Restelli. Transfer of task representation in reinforcement learning using policy-based proto-value functions (short paper). In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 1329–1332, 2008.

L. Frommberger. Learning to behave in space: A qualitative spatial representation for robot navigation with reinforcement learning. *International Journal on Artificial Intelligence Tools*, 17(3): 465–482, 2008.

A. Guazzelli, F.J. Corbacho, M. Bota, and M.A. Arbib. Affordances, motivations, and the world graph theory. *Adaptive Behavior*, 6(3/4):433–471, 1998.

B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250, 2002.

A. Jonsson and A.G. Barto. Automated state abstraction for options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems 13*, pages 1054–1060, 2001.

A. Jonsson and A.G. Barto. A causal approach to hierarchical decomposition of factored MDPs. In *Proceedings of the Twenty Second International Conference on Machine Learning*, 2005.

G.D. Konidaris and G.M. Hayes. Estimating future reward in reinforcement learning animats using associative learning. In *From Animals to Animats 8: Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior*, pages 297–304, July 2004.

A. Laud and G. DeJong. The influence of reward on the speed of reinforcement learning: an analysis of shaping. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 440–447, 2003.

A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*, pages 544–551, 2008.

S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty First International Conference on Machine Learning*, pages 560–567, 2004.

M.J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997.

A. McGovern and A.G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368, 2001.

N. Mehta, S. Ray, P. Tadepalli, and T. Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the Twenty Fifth International Conference on Machine Learning*, 2008.

A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

A.Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, 1999.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049, 1997.

T.J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, Department of Computer Science, University of Massachusetts, Amherst, 1999.

M. Pickett and A.G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference of Machine Learning*, pages 506–513, 2002.

D. Precup, R.S. Sutton, and S. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 759–766, 2000.

M.L. Puterman. *Markov Decision Processes*. Wiley, 1994.

J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pages 463–471, 1998.

B. Ravindran and A.G. Barto. Relativized options: Choosing the right transformation. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 608–615, 2003a.

B. Ravindran and A.G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi markov decision processes. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1011–1016, 2003b.

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

Ö. Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758, 2004.

Ö. Şimşek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, 2005.

S. Singh, A.G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *Proceedings of the 18th Annual Conference on Neural Information Processing Systems*, 2004.

B. F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. Appleton-Century-Crofts, New York, 1938.

M. Snel and S. Whiteson. Multi-task evolutionary shaping without pre-specified representations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1031–1038, 2010.

P. Stone, R.S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

R.S. Sutton, D. Precup, and S.P. Singh. Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 556–564, 1998.

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

M.E. Taylor and P. Stone. Value functions for RL-based behavior transfer: a comparative study. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, 2005.

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

M.E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems*, 2008.

S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press, 1995.

L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Skill acquisition via transfer learning and advice taking. In *Proceedings of the Seventeenth European Conference on Machine Learning*, pages 425–436, 2006.

J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2000.

E. Wiewiora. Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

E. Wiewiora, G. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 792–799, 2003.

A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 1015–1022, 2007.

W. Zhang and T.G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence*, pages 1114–1120, 1995.