Using Homomorphisms to Transfer Options across Continuous Reinforcement Learning Domains

Vishal Soni and Satinder Singh
Computer Science and Engineering
University of Michigan, Ann Arbor
{soniv, baveja}@umich.edu

Abstract

We examine the problem of *Transfer* in Reinforcement Learning and present a method to utilize knowledge acquired in one Markov Decision Process (MDP) to bootstrap learning in a more complex but related MDP. We build on work in *model minimization* in Reinforcement Learning to define relationships between state-action pairs of the two MDPs. Our main contribution in this work is to provide a way to compactly represent such mappings using relationships between state variables in the two domains. We use these functions to transfer a learned policy in the first domain into an *option* in the new domain, and apply intra-option learning methods to bootstrap learning in the new domain. We first evaluate our approach in the well known Blocksworld domain. We then demonstrate that our approach to transfer is viable in a complex domain with a continuous state space by evaluating it in the Robosoccer Keepaway domain.

Introduction

Transfer learning involves taking knowledge acquired in one domain, the *source domain*, and utilizing it in a new but possibly similar domain, the *target domain*. In this paper, we examine transfer learning in the context of Markov Decision Processes (MDPs). Specifically, we consider transfer scenarios in which the agent first learns a policy in one MDP and then applies that knowledge towards learning in a more complex (in the size of the state and action space) MDP. The potential benefit we seek is that the transferred knowledge will help bootstrap the learning process in the target domain.

We build on the framework of MDP Homomorphisms (Ravindran & Barto 2002). Homomorphisms are an abstraction framework that allow us to define transformations between MDPs based on their transition dynamics and reward functions. Any homomorphism between two MDPs allows us to take a policy from one MDP and transfer it to the other. So far, the use of Homomorphisms for transfer learning in MDPs is limited because (a) Homomorphisms require a strict correspondence in rewards between the target and source MDPs, (b) Homomorphisms are difficult to learn (Ravindran & Barto 2003), and (c) Homomorphisms are hard to specify as prior knowledge, especially in domains with continuous state spaces.

In this paper, our main contribution is a way to compactly represent a set of transformations between the target and source MDPs by defining relationships between their state variables, thus allowing us to easily specify such transformations. We also show that by not requiring rewards in the target and source domains to correspond exactly, we are able to generate a larger set of potentially useful transformations.

We evaluate our approach in two test domains - the Blocksworld domain and the Robosoccer Keepaway domain. The Blocksworld domain consists of a set of differently shaped blocks. The task of the learner is to take the blocks from some initial configuration and rearrange them into some final configuration. Our transfer task in this domain is to take a policy for rearranging a certain number of blocks and apply it towards rearranging a larger number of blocks. The Keepaway domain is a fairly complex domain with a continuous state space and consists of a team of *keepers* who endeavor to keep a ball away from a team of *takers* for as long as possible. Our transfer task in this domain is to take a policy to play three-versus-two Keepaway (three keepers and two takers) and to use it to bootstrap learning in four-versus-three Keepaway (four keepers and three takers).

Before describing our contribution, we provide background on the three components of this work: Markov Decision Processes, Options, and Homomorphisms.

Preliminaries

Markov Decision Processes

A Markov Decision Process (MDP) is a tuple of $\langle S, A, P, R\rangle$ where S is the set of states, A is the set of actions, P is the transition probability function with $P(s, a, s')$ being the probability of transitioning from state s to state s' under action a. R is the reward function with $R(s, a)$ denoting the reward for performing action a in state s. The notation $A(s)$ refers to a set of admissible actions in state s.

A policy $\pi : S, A \rightarrow [0, 1]$ defines the behavior of an agent in its environment. The value of a policy π in any state s, denoted $V^{\pi}(s)$, is the expected return when starting in state s and following the policy π thereafter. The solution to an MDP is an optimal policy π^* such that for all $s \in S$, $V^{\pi^*}(s) \geq V^{\pi}(s)$ for any policy π. Reinforcement Learning provides several algorithms which approximate the optimal value function in an MDP.

Often when dealing with large or continuous state spaces, an MDP is represented in a structured form in terms of its state variables. In such representations, called factored representations (e.g. Guestrin *et al.* (2003)), the state space S is defined by a set of k state variables. A state $s \in S$ is is a unique assignment $s = \{s_1, \cdots, s_k\}$ to each of these state variables. The set of permissible values for any state variable is known as its *domain*. Transition probabilities and rewards are then also represented in a factored form in terms of state variables.

Options

At a very high level, an option (Sutton, Precup, & Singh 1999) can be thought of as a sub-routine, or skill, that the agent can invoke in certain states. Unlike primitive actions that last for one time step, an option can span multiple time steps. We use options to represent the agent's transferred knowledge in the target MDP.

An option in an MDP $M = \langle S, A, P, R \rangle$ is defined by the tuple $O = \langle \mathcal{I}, \pi^O, \beta \rangle$ where $\mathcal{I} \subseteq S$ is the set of states in which the option can be invoked. The policy π^O specifies the option's behavior while the option is being executed, and β specifies a set of termination conditions that indicate when the option terminates. A primitive action $a \in A$ can be thought of as a special case of an option that lasts for exactly one time step and whose availability is defined by the availability of a. A set of options $\mathcal{O}$ can thus contain multi-step options as well as primitive actions defined as one-step options. Finally, whereas previously we defined policies over state-action pairs, we can now define policies over state-option pairs, i.e. $\pi : S \times \mathcal{O} \rightarrow [0, 1]$.

While it may seem that having to learn a policy over options in addition to primitive actions can negatively effect learning complexity, this needn't be the case. If the option policies are Markov and if we have access to the option policy, we can employ powerful learning methods called *Intra-option learning* (Sutton, Precup, & Singh 1998) that allow us to update the value of an option while behaving according to another option. These methods can thus be applied to learn simultaneously about many options from the same experience and can greatly speed up the learning process.

Homomorphisms

Formally, a Morphism of a structured system is some transformation of the system that preserves its structure. MDP Homomorphisms are an approach to abstraction in MDPs and are part of a class of algorithms known as model minimization (Dean & Givan 1997) methods for MDPs.

In general, model minimization techniques divide the state space of an MDP into disjoint subsets, or *blocks*, of states and define a new reduced MDP over these blocks. A *partition* $\mathbf{B}$ of states S is a collection of blocks $\mathbf{b_i} \subseteq S$ such that $\cup_i \mathbf{b_i} = S$. The block-action transition probabilities are given by a function T where $T(\mathbf{b}, a, \mathbf{b}')$ is the probability that when action a is taken in any state $s \in \mathbf{b}$, the next state is in $\mathbf{b}'$. That is, for all $\mathbf{b}, \mathbf{b}' \in \mathbf{B}$ and for all $s \in \mathbf{b}$,

$$T(\mathbf{b}, a, \mathbf{b}') = \sum_{t \in \mathbf{b}'} Pr(s, a, t)$$

Therefore, for any action a, all states $s \in \mathbf{b}$ have the same probability of transitioning into the block $\mathbf{b}'$.

A homomorphism between a target MDP M and a source MDP M' is a function h that maps block-action pairs in M to state-action pairs in M' such that (a) block-action transition probabilities in M are equal to the corresponding state-action transition probabilities in M', and (b) the reward associated with all the states in a particular block in M is the same as the reward associated with the corresponding state in M'.

The function h is written in a modular form as a tuple of surjections $h = \langle f, g_s | s \in S \rangle$ where $f : S \rightarrow S'$ maps states in each block in M to a unique state in M', and $g_s : A_s \rightarrow A'_{f(s)}$ maps admissible actions in state s to admissible actions in state $f(s)$.

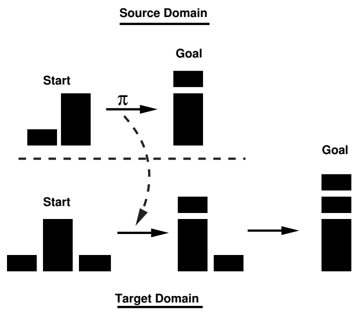


Figure 1: A simple example of transfer in Blocksworld where a policy learned in the source domain (above) cannot be transferred to attain the goal in the target domain (below) and, as such, no homomorphism exists between the two domains. However, if we relax the homomorphism constraint on rewards, the policy π in the source can be transferred to attain a useful subgoal (below middle) in the target.

Transfer Learning with Homomorphisms

Homomorphisms provide an ideal framework for transfer from a less complex MDP to a more complex one. Ravindran & Barto (2002) show that given a stochastic policy π' in a source MDP M', a homomorphism $h = \langle f, g_s | s \in S \rangle$ from target M to source M' can be used to induce a stochastic policy π in the target MDP M. Let $g_s^{-1}(a') \subseteq A_s$ denote the set of target actions that map to the same source action $a' \in A'_{f(s)}$. Then for any $s \in S$ and $a \in g_s^{-1}(a')$,

$$\pi(s, a) = \frac{\pi'(f(s), a')}{| \ g_s^{-1}(a') \ |}$$

That is, the probability of taking action a in target state s is the probability of taking action a' in the source state s' normalized by the number of actions that map to a' in state s.

In fact, if π' is optimal in M', Ravindran & Barto (2002) show that the induced policy π is optimal in M.

```
Initialize $\vec{\theta}$ arbitrarily
Repeat (for each episode):
    $\vec{e} = \vec{0}$
    $s, o \leftarrow$ initial state, option of episode
    $a \leftarrow \pi_o(s)$
    For all options $o' \in \mathcal{O}(s)$:
        $\mathcal{F}_{s,o'} \leftarrow$ set of features present in $s, o'$

    Repeat (for each step of the episode):
        # update eligibility trace
        $\forall o' \in \mathcal{O}(s)$ s.t. $\pi_{o'}(s) = a$,
            $\forall i \in \mathcal{F}_{s,o'} : e(i) \leftarrow e(i) + 1$

        $Q(s,o) \leftarrow \sum_{i \in \mathcal{F}_{s,o}} \theta(i)$
        Take action $a$, observe reward $r$, next state $s'$

        # update q values
        For all options $o' \in \mathcal{O}(s')$:
            $\mathcal{F}_{s',o'} \leftarrow$ set of features present in $s', o'$
            $Q(s',o') \leftarrow \sum_{i \in \mathcal{F}_{s',o'}} \theta(i)$
        $o^* \leftarrow \arg\max_{o'} Q(s',o')$
        For all options $o' \in \mathcal{O}(s')$:
            $q \leftarrow Q(s',o')(1 - \beta_{o'}(s)) + Q(s',o^*)\beta_{o'}(s)$
            $\delta \leftarrow r - Q(s,o') + \gamma q$
            $\vec{k} = \vec{e}$
            $\forall i \in \mathcal{F}_{s,o'} : k(i) \leftarrow \delta k(i)$
        $\vec{\theta} \leftarrow \vec{\theta} + \alpha \vec{k}$

        # decay eligibility trace
        For all options $o' \in \mathcal{O}(s')$:
            $\mathcal{F}_{s',o'} \leftarrow$ set of features present in $s', o'$
            $Q(s',o') \leftarrow \sum_{i \in \mathcal{F}_{s',o'}} \theta(i)$
        $o^* \leftarrow \arg\max_{o'} Q(s',o')$
        $\vec{e} \leftarrow \gamma\lambda\vec{e}$
        With probability $1 - \epsilon$
            $o = o^*$
        else
            $o \leftarrow$ random option in $\mathcal{O}(s')$
            if $\pi_o(s') \neq \pi_{o^*}(s')$
                $\vec{e} \leftarrow \vec{0}$
        $s \leftarrow s', a \leftarrow \pi_o(s')$

    until $s$ is terminal
```

Figure 2: Intra-option Q-learning with function approximation to learn policy over options as well as primitive actions.

Homomorphisms can also be used to induce options onto the target MDP. In their work on relativised options, Ravindran & Barto (2003) show that given an option $O' = \langle \mathcal{I}', \pi^{O'}, \beta' \rangle$ in a source MDP M', a homomorphism h from target M to source M' induces an option $O = \langle \mathcal{I}, \pi^{O}, \beta \rangle$ in the following manner:

- The initiation set of the option O is the set of states $s \in f^{-1}(s'), \forall s' \in \mathcal{I}'$
- The option policy is a policy π^O induced by the homomorphism h given the option policy $\pi^{O'}$ (analogous to the manner described above)
- The termination condition in any state $s \in S$ is $\beta(s) = \beta'(f(s))$, i.e. defined by the termination condition of the source option.

Since discovering Homomorphisms is NP-hard (Ravindran & Barto 2003), current approaches to learning with Homomorphisms involve supplying the agent with a set of candidate transformations and a policy learned in the source MDP. The agent then learns the best homomorphism to apply in any state of the target MDP.

Generalizing Transfer Over Target Reward Functions

From a transfer perspective, a limitation of using Homomorphisms is that the rewards in the target and source MDPs have to correspond, i.e. the target states that correspond to the reward state in the source MDP must also have the same reward associated with them.

For the kinds of problems motivating this work, we expect that rather than being directly applicable to accomplishing the task specified in the target MDP, the transferred knowledge may provide a useful way of behaving as a step towards accomplishing the target task. Figure 1 provides an illustration of this in the Blocksworld domain. To extricate ourselves from these reward structure constraints, we partition the state-action space of the target MDP based solely on transition probabilities (and ignore the reward constraint). We are thus not constrained to having an exact mapping between reward functions in the two domains. The more similar the reward functions are, however, the more useful some subset of induced options will be in optimizing the target reward.

Of course, we do not know which of these options will be useful in the target MDP. In our experiments, we examine whether our learning algorithm (Figure 2) can quickly learn to exploit the more useful options (and hence benefit from similarities in reward structures of the two MDPs) while ignoring the less useful ones.

Generating Candidate transformations

As mentioned earlier, approaches to learning with Homomorphisms involve supplying the agent with a set of candidate transformations, which involves specifying a set of tuples $\langle f, g_s \rangle$. However, in continuous domains it can be hard, or even impossible, to supply the function f between states in the target and source domains.

Instead, we map *state variables* in the target domain to state variables in the source domain. We consider factored MDPs and provide the agent with a tuple $\langle X, g_s \rangle$, where X is a surjection that maps state variables in the target domain to state variables in the source domain. We now show that the tuple $\langle X, g_s \rangle$ can used to generate a set of transformations $\langle f, g_s \rangle$. The benefit is that in continuous domains, if the state can be factored into a finite set of state variables, the variable mapping function X will be easier to specify than the state mapping function f.

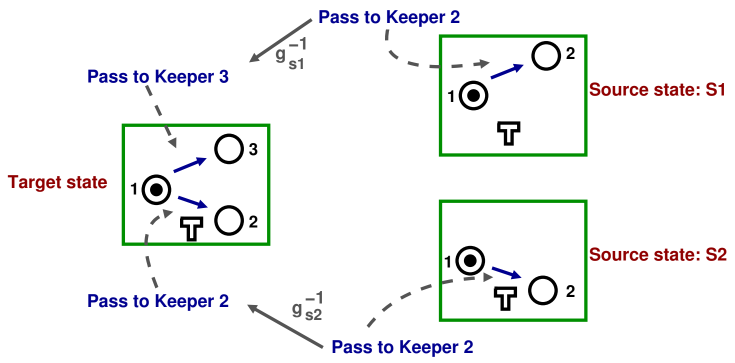


Figure 3: A simple example of transfer from the source domain, 2v1 Keepaway (right) to the target domain, 3v1 Keepaway (left). The state variable mapping, X, maps the variables *distance to keeper 2* and *distance to keeper 3* in the target domain to the variable *distance to keeper 2* in the source domain. This figure shows the resulting state mapping for a particular target state. The solid arrows indicate the actions available to the keeper with the ball (concentric circle). The taker, 'T', can intercept one of those passes. Dotted lines represent the action mapping function g_s. So an action *Pass to 2* from the top right state will map to *Pass to 3* in the target state.

Of course, since the function g_s is provided, we have only generate a set of functions f in order to generate a set of candidate transformations.

Formally, define X to be a surjection that maps variables in the target MDP M to variables in the source MDP M', i.e. $X : V \rightarrow \{V' \cup \nu\}$, where ν is "dummy" variable. By allowing some variables in V to map to a dummy variable, we can define a partial mapping between state variables. That is, we are not restricted to mapping every single target variable to some source variable.

Any one to one mapping between variables in the target M and the source M' will give us a single function f. Since X is a many to one mapping from variables in the target to variables in the source, any projection of this mapping onto a one-to-one mapping between state variables will generate a function f. Since multiple such projections are possible, multiple state mapping functions f can be generated. The function X thus represents multiple f functions and the tuple $\langle X, g_s\rangle$ represents a set of candidate transformations. Figure 3 demonstrates this for a simple Keepaway problem.

The advantage for continuous state spaces is that we do not have to store a set of candidate transformations. We only need store the tuple $\langle X, g_s\rangle$, which is easier to do if have a finite number of actions and state variables.

Note that more generally, we can provide a set of surjections $X \in \mathcal{X}$ with each element in the set representing multiple f functions.

Learning Algorithm Since we do not explicitly store candidate transformations, it is not possible to explicitly induce options. However, given an $\langle X, g_s\rangle$, we can implicitly represent an option O_i as the tuple $\langle i, g_s, \pi'\rangle$ where i represents the i^{th} state mapping function f_i, and π' is a policy in the source. The option O_i can be defined in terms of its initiation set, policy, and termination condition as follows:

- The initiation set of O_i is all the states in which the i^{th} transformation is possible.
- Let $g_s^{-1}(a') \subseteq A_s$ denote the set of target actions that map to the same source action $a' \in A'_{f(s)}$. If π' is the policy in the source MDP M', then for any $s \in S$ and $a \in g_s^{-1}(a')$ the option policy π^{O_i} is given by
$$\pi^{O_i}(s,a) = \frac{\pi'(f_i(s), a')}{\mid g_s^{-1}(a') \mid}$$
where f_i is the i_{th} state mapping function under X.
- The option terminates in all the states in which the i^{th} transformation is not possible, or when the source policy π' terminates.

The algorithm present in Figure 2 employs Q-learning with CMAC function approximation (Sutton & Barto 1998) and intra-option learning to learn a value function over these implicit options, effectively learning which transformation to pick in any given state.

Experiments

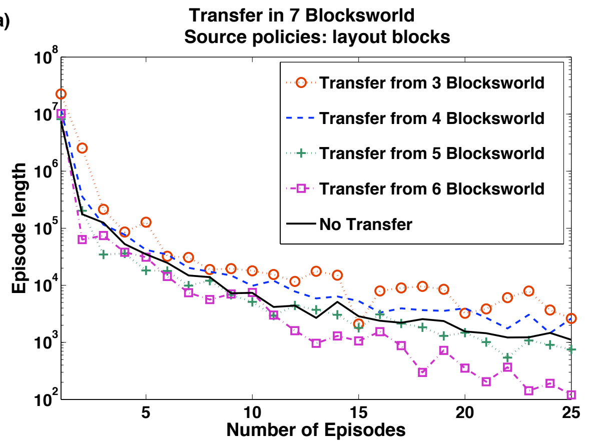
We now present an evaluation of our approach on the Blocksworld and the Robosoccer Keepaway domains.

Blocksworld consists of a set of distinctly labeled objects that can either be stacked on each other or placed on a table. The task of the agent is to take the objects from an initial configuration and rearrange them to arrive at the goal configuration. The agent can only move one object at a time, and at most one object can reside on top of another. An object can be moved to any object that is clear (i.e. has no other object on top of it) or to the table. In a world with N objects, the state is determined by a set of binary state variables - an "on" value for binary variable $i_o j$ indicates that object i is over object j. An action $a_{ij}, i \in [1, N], j \in \{[1, N] \cup \mathit{table}\}, i \neq j$, moves object i on top of object j. The action a_{ij} is available in all states in which both i and j are clear (the table is always clear).

An object i can have one of three different shapes which determines the probability with which it can be successfully placed on top of another object. So taking action a_{ij} will result in object i residing on top of object j with some probability p_{ij}, and falling on the table with probability $1 - p_{ij}$. Finally, we introduce some noise in the system such that the transition dynamics of any two objects of the same shape differ by some ϵ, i.e. if objects i and k have the same shape, $p_{ij} = p_{kj} \pm \epsilon$. Similarly, $p_{ji} = p_{jk} \pm \epsilon$.

In all of our tests the target domain was the 7-Blocksworld domain which contained 7 objects of different sizes. We used four source domains, the domains from which a policy was transferred. There were 3-Blocksworld, 4-Blocksworld, 5-Blocksworld, and 6-Blocksworld. We defined a set of inter-domain variable mapping functions X by mapping state variables between domains based on the shapes of the objects they represent. So a binary target variable $i_o j$ maps to a source variable $z_o x$ if i and z, and j and x have the same

(a)



(b)

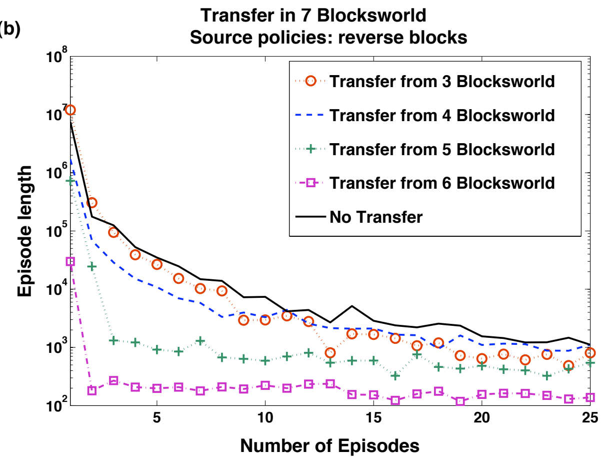


Figure 4: Results of transfer to 7-Blocksworld from 3,4,5, and 6 Blocksworlds. In (a) the tasks in the source domains were to lay the objects out flat on the table. In (b) tasks all the other Blocksworld were also to reverse the objects. Each plot line represents results averaged over thirty runs.

shape. Note that this mapping represents a set of the variable mapping functions defined in the previous section. The probabilities of stacking a block of a particular shape on another block were randomly generated in the range $[0.6, 0.9]$.

In all our experiments, the start state in all domains was one in which objects were stacked in order of their label. So block-1 was over block-2, and so on. The goal state for the target domain (7-Blocksworld) was to stack the blocks in reverse order of their label. We learned two different policies in each source domain and evaluated how well these policies transferred to achieving the goal in the target domain.

In Figure 4(a), the goal state for all the source domains was a configuration in which objects were laid out flat on the table. These policies were then used to bootstrap learning in the 7-Blocksworld. While the tasks in source and target domains are different, we expect to see some benefit from transfer since laying out objects flat on the table is most likely en-route to assembling them in reverse order. As we would expect, the most benefit comes from the policy that was transferred from 6-Blocksworld. However, it is interesting to see that the policies transferred from 3-Blocksworld and 4-Blocksworld caused the learner to perform worse than the from-scratch learner (no transfer). It seems that the policies transferred from 3-Blocksworld and 4-Blocksworld are too distracting. One possible explanation for this is that these policies are available in a lot more states than 6-Blocksworld or 5-Blocksworld. It may be the case that in many of those states, attempting to lay objects out on the table is not a useful course of actions.

In Figure 4(b), the goal state for all the source domains was to reverse the ordering of the objects (just like in the target domain). We see that when the policies from the source domains are transferred to 7-Blocksworld, they have a positive impact on learning performance. This is expected since the reward structures in the source and target domains are very similar. Comparing 4(a) to 4(b), we see that the learning algorithm is able to extract greater benefit from the transferred policies when the reward structures in the source and target domains are more similar.

Keepaway is a sub-problem of the Robosoccer domain in which one team, the *Keepers*, endeavors to keep possession of a ball and the opponents, the *takers*, attempt to gain possession of the ball. Whenever a taker acquires the ball, or when the ball exits the playing field, an episode terminates and a new one begins. A game is characterized by the size of the playing field, the number of keepers and the number of takers. Stone, Sutton, & Kuhlmann (2005) map the Keepaway problem into a discrete time, episodic reinforcement learning framework, which we utilize here.

State variables are comprised of an agent's distance and angle relative to other players. Keepers make a decision only when they possess the ball. The actions available to a keeper with the ball are *Pass Ball to Keeper i* and *Hold Ball*. Keepers without the ball follow a deterministic policy - the keeper closest to the ball moves to acquire the ball while the others move to an open region so as to be able to receive a pass. Takers also follow a deterministic policy of moving to passing lanes so as to be able to intercept a pass. Random noise is injected into all observations and actions. Communication is not allowed between agents except through the simulator and thus agents cannot share their knowledge.

To define the set of variable-mapping functions, we mapped all the variables corresponding to keeper (taker) positions in 4v3 Keepaway to the variables corresponding to keeper (taker) positions in 3v2 Keepaway. From a high level, this is essentially telling the agent who its teammates and opponents are and, as far as giving prior domain information is concerned, is a fairly reasonable type of knowledge to provide. In fact, this information is already encoded in the actions (since keepers can only pass the ball to keepers).

This is a significantly more complex domain than Blocksworld for several reasons not the least of which is that this a multi-agent system with a continuous state space. The possibility of transfer in this domain has been explored by Taylor & Stone (2005) where they use a specific transformation to transfer learned knowledge (as opposed to our approach of generating a set of transformations). We now

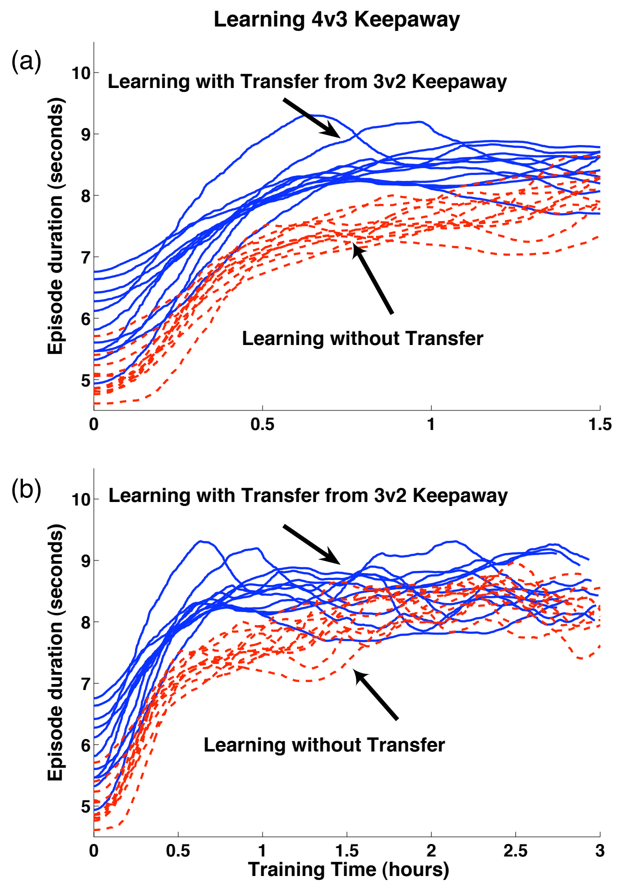


Figure 5: Learning 4v3. The solid blue lines show the performance of the Keepers that transferred a policy learned in 3v2. Note that (a) and (b) are the same plot, but displayed over different time scales.

examine whether the approach based on MDP Homomorphisms is relevant to transfer in this domain. We first trained keepers in a 3v2 (3 keepers versus 2 takers) domain. We then used the policies learned here to bootstrap learning in the 4v3 domain and compared learning performance against keepers learning 4v3 Keepaway from scratch. We measured effectiveness of transfer by how much sooner the team of keepers bootstrapping with transferred knowledge were able to achieve a target hold time of 8 seconds versus the team of keepers learning without any transferred knowledge.

Figure 5 presents our results. The y-axis indicates episode length - longer episodes imply better keeper performance. Each plot line represents a single trial over a three hour training period and is smoothed by averaging over a window of a hundred episodes. In Figure 5(a), keepers with transferred knowledge attained the target about 30 minutes sooner than keepers without transferred knowledge. Figure 5(b) plots the same data but on a longer time scale. We see that in the long term, keepers that bootstrap with transferred knowledge perform as well as keepers learning from scratch indicating that these transferred policies, which may not be optimal in the target domain, do not hinder performance in the long term. We are thus indeed able to bootstrap learning in 4v3 Keepaway by transferring a policy learned in 3v2 Keepaway without loss of performance in the long term.

Conclusions

We have examined the problem of transfer in MDPs. We showed that by defining a mapping between state variables, we can compactly represent a set of candidate transformations between a target and a source MDP. We showed that intra-option learning methods can be used to determine which transformation to apply in any given state. By considering transformations based solely on transition probabilities, we are able to transfer policies from the source domain to a larger set of options in the target domain. We evaluated our approach in the Blocksworld and the Robosoccer Keepaway domains and showed that this is a viable approach to transfer in continuous domains.

Acknowledgments

This work is supported by the NSF under Grant Number CCF-0432027 and by a grant from DARPA's IPTO program. 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 NSF or DARPA.

References

Dean, T., and Givan, R. 1997. Model minimization in markov decision processes. In *American Association for Artificial Intelligence*, 106 – 111.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19:399–468.

Ravindran, B., and Barto, A. G. 2002. Model minimization in hierarchical reinforcement learning. In *Fifth Symposium on Abstraction, Reformulation and Approximation*, 196–211.

Ravindran, B., and Barto, A. G. 2003. An algebraic approach to abstraction in reinforcement learning. In *Twelfth Yale Workshop on Adaptive and Learning Systems*, 109–144. Yale University.

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

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

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

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

Taylor, M. E., and Stone, P. 2005. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 53–59.