Skill Acquisition Via Transfer Learning and Advice Taking*

Lisa Torrey[1], Jude Shavlik[1], Trevor Walker[1], and Richard Maclin[2]

[1] University of Wisconsin, Madison WI 53706, USA
[2] University of Minnesota, Duluth, MN 55812, USA

Abstract. We describe a reinforcement learning system that transfers skills from a previously learned source task to a related target task. The system uses inductive logic programming to analyze experience in the source task, and transfers rules for when to take actions. The target task learner accepts these rules through an advice-taking algorithm, which allows learners to benefit from outside guidance that may be imperfect. Our system accepts a human-provided mapping, which specifies the similarities between the source and target tasks and may also include advice about the differences between them. Using three tasks in the RoboCup simulated soccer domain, we demonstrate that this system can speed up reinforcement learning substantially.

1 Introduction

Machine learning tasks are often addressed independently, under the implicit assumption that each new task has no relation to the tasks that came before. In some domains, particularly reinforcement learning (RL) ones, this assumption is often incorrect since tasks in the same domain tend to be related. Even tasks that are quite different in their specifics may have general similarities, such as shared *skills*; that is, conditions under which an agent should take an action. Our goal is to *transfer* general skills from a source task in order to speed up learning in a new but similar target task.

For example, suppose an RL soccer player has learned, in a source task, to keep the ball from its opponents by passing to its teammates. In the target task, suppose it must learn to work with teammates to score goals against opponents. If this player could apply its passing skills from the source task, it might master the target task more quickly.

Even when RL tasks have shared skills, transfer between them is a difficult problem because differences in action sets and reward structures create differences in shared skills. For example, the passing skill in the source task above is incomplete for the target task, where passing needs to cause progress toward the goal. This indicates that RL agents using transferred information must continue to learn, filling in gaps left by transfer. Since transfer might also produce

* This research is partially supported by DARPA grant HR0011-04-1-0007 and US Naval Research Laboratory grant N00173-06-1-G002.

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): ECML 2006, LNAI 4212, pp. 425–436, 2006.

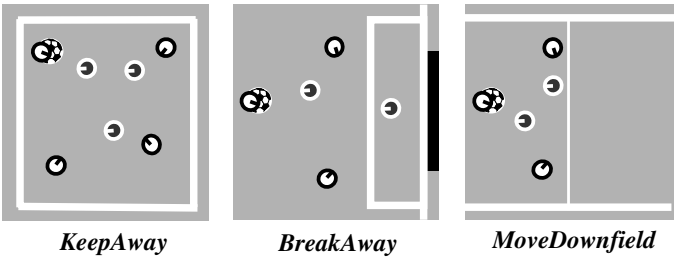

Fig. 1. Snapshots of RoboCup soccer tasks

partially irrelevant or incorrect skills, RL agents must also be able to modify or ignore transferred information that is imperfect.

One way to facilitate transfer is for a human observer with basic domain knowledge to provide a *mapping* between source and target tasks. A mapping describes the structural similarities between the tasks, such as correspondences between player objects in the example above. It might also include simple advice that reflects the differences between the tasks. In our example, tips like "prefer passing toward the goal" and "shoot when close to the goal" would be helpful.

We present a system for transfer learning in RL called AI^2 (Advice via Induction and Instruction). It constructs relational *transfer advice* by using inductive logic programming to analyze experience in the source task and learn skills in first-order logic. The user contributes a mapping between the tasks that may include *user advice*. The target-task learner considers the advice while learning and can follow it, refine it, or ignore it according to its value.

The AI^2 approach performs transfer at a higher level of abstraction than some previous approaches [14,15]. For this reason, it performs well in transfer scenarios involving more distant tasks. We present empirical results in the challenging RoboCup simulated soccer domain, demonstrating significantly faster learning in the target task *BreakAway* [15] after performing user-guided transfer from the source tasks *KeepAway* [9] and *MoveDownfield* (see Figure 1).

2 Reinforcement Learning in RoboCup

In reinforcement learning [13], an agent navigates through an environment trying to earn rewards or avoid penalties. The environment's state is described by a finite number of features, and the agent takes actions to cause the state to change. In Q-learning, the agent learns a Q-function to estimate the value of taking an action from a state. An agent's policy is typically to take the action with the highest Q-value in the current state, except for occasional exploratory actions. After taking the action and receiving some reward, the agent updates its Q-value estimates for the current state. AI^2 uses the $SARSA$ and TD(λ) reinforcement learning algorithms designed by Sutton [11,12].

In the RoboCup learning task of M-on-N KeepAway [9], the objective of the M reinforcement learners called *keepers* is to keep the ball away from N

hand-coded players called *takers*. The game ends when an opponent takes the ball or when the ball goes out of bounds. The learners receive a +1 reward for each time step their team keeps the ball. Keepers without the ball follow a hand-coded strategy to receive passes.

In the original KeepAway task, the keeper who has the ball can choose only to hold it or pass to a teammate. We introduce a new version called *Mobile KeepAway*, in which this keeper can also move (inwards, outwards, clockwise and counterclockwise with respect to the field center). With more realistic movement, this version may transfer better to other games.

Our KeepAway state representation is based on the one designed by Stone and Sutton [9]. The keepers are ordered by their distance to the learner *k0*, as are the takers. The features are listed in Table 1.

Note that our logical variables are capitalized and typed (*Player*, *Keeper*, etc.). For simplicity we indicate types by variable names, leaving out terms like *player(Player)*, *keeper(Keeper)*, etc. Constants are uncapitalized.

A second RoboCup task is M-on-N BreakAway [15], where the objective of the M reinforcement learners called *attackers* is to score a goal against $N - 1$ hand-coded *defenders* and a hand-coded *goalie*. The game ends when they succeed, when an opponent takes the ball, when the ball goes out of bounds, or after a time limit of 10 seconds. The learners receive a +1 reward if they score a goal, and zero reward otherwise. Attackers without the ball follow a hand-coded strategy to receive passes. The attacker who has the ball may choose to move (ahead, away, left, or right with respect to the goal), pass to a teammate, or shoot (at the left, right, or center part of the goal).

Our BreakAway state representation is the one presented in Torrey et al. [15]. The attackers are ordered by their distance to the learner *a0*, as are the defenders. The features are listed in Table 1.

We also introduce a third RoboCup task called M-on-N MoveDownfield, where the objective of the attackers is to move toward the opposing team's goal while maintaining possession of the ball. The game ends when they cross a vertical line on the field, when an opponent takes the ball, when the ball goes out

Table 1. RoboCup task feature spaces

KeepAway features	*BreakAway and MoveDownfield features*
distBetween(k0, Player) distBetween(Keeper, ClosestTaker) angleDefinedBy(Keeper, k0, ClosestTaker) xPosition(Object) yPosition(Object) distBetween(Keeper, fieldCenter)	distBetween(a0, Player) distBetween(Attacker, ClosestDefender) angleDefinedBy(Attacker, a0, ClosestDefender) xPosition(Object) yPosition(Object) distBetween(Attacker, goalCenter) distBetween(a0, GoalPart) angleDefinedBy(GoalPart, a0, goalie) angleDefinedBy(topRight, goalCenter, a0) distBetween(Attacker, goalie) angleDefinedBy(Attacker, a0, goalie) timeLeft

of bounds, or after a time limit of 25 seconds. The learners receive symmetrical positive and negative rewards for horizontal movement forward and backward. Attackers without the ball follow a hand-coded strategy to receive passes. The action set and feature set are the same as in BreakAway, except without the *shoot* actions and most of the features involving the goal.

Our system discretizes each feature in these tasks into 32 intervals called *tiles*, each of which is associated with a Boolean feature. For example, the tile denoted by $distBetween(a0, a1)_{[10,20]}$ takes value 1 when *a1* is between 10 and 20 units away from *a0* and 0 otherwise. This enhancement of the state space is used in RoboCup by Stone and Sutton [9], and we adopt it to give our linear Q-function model the ability to represent more complex functions.

These three RoboCup games have substantial differences in features, actions, and rewards (and therefore Q-values), but they all require the skill of passing the ball among teammates without losing it to the opponents.

3 AI^2: Transferring Skills

Because RL agents learn to take actions, a natural human interpretation of RL is that the agents acquire *skills*. However, what they typically acquire is a Q-function, which is highly task-specific and does not readily translate into discrete skills. Some researchers have therefore proposed methods for transferring an entire Q-function or policy [14,15].

We present an approach that does not use the Q-function to perform transfer. Instead, it analyzes games played in the source task to learn *skills* in first-order logic. By learning high-level concepts, AI^2 favors the transfer of general, behavioral information over the specific, low-level details of the Q-function.

In the AI^2 framework, games are collections of state-action pairs where the action is the classification of the state. It uses these pairs as training examples to learn to classify states. For example, from traces of KeepAway games, AI^2 can learn the concept "states in which passing to a teammate is a good action."

AI^2 can be used when a new task arises in a domain and data from an old task already exists. To use it, the user identifies which skills should be transferred, provides a mapping that relates logical objects in the source task to those in the target task, and optionally gives advice about new or transferred skills.

Table 2. The AI^2 algorithm

GIVEN	DO
Game traces from source task	For each skill to transfer:
List of skills to be transferred	Collect training examples
Object mapping between tasks	Learn rules with Aleph
User advice (optional)	Select rule with highest $F(\beta)$ score
	Translate rule into transfer advice
	Learn target task with all advice

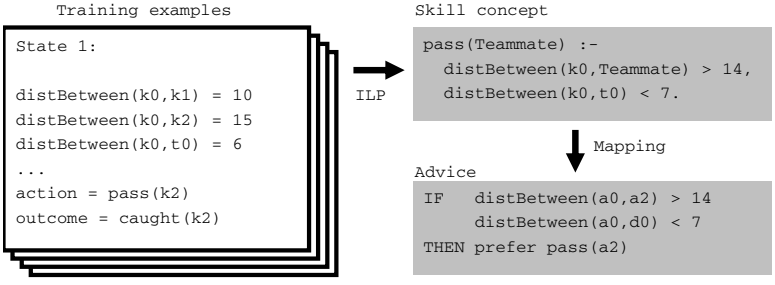

Fig. 2. Example showing how AI^2 transfers skills

Given this information, AI^2 performs transfer automatically. From existing game traces in the source task, the system learns skill concepts and translates them into advice for the target task. It then applies both the transfer advice and the user advice to learning in the target task.

Table 2 summarizes the AI^2 algorithm in high-level pseudocode. Figure 2 illustrates the transfer part of this algorithm with an example from RoboCup.

Each advice item is a conjunction of conditions and a constraint to be applied if the conditions are met, as shown in Figure 2. Advice need not be followed exactly; it can be refined or even ignored if it disagrees with the learner's experience, using the advice-taking algorithm of Maclin et al. [5], which we explain in Section 4. As we demonstrate with an experiment in Section 5, this provides some protection against imperfect transfer.

3.1 Learning Skills

AI^2 uses inductive logic programming (ILP) to learn skills. ILP is a method for learning first-order conjunctive rules that works with data described by logical relations, such as the RoboCup feature space as presented in Section 2.

A first-order rule, unlike a propositional rule, can contain variables like *Teammate* in Figure 2. The advantage of first-order rules is that they are more general. For example, the rule *pass(Teammate)* is likely to capture the essential elements of the passing skill better than rules for passing to specific teammates. We expect these common skill elements to transfer better to new tasks.

An advantage of ILP in general is that it can accommodate background knowledge for a domain. Our system allows a sophisticated user to define new predicates and add them to the search space. For example, we added predicates to the RoboCup domain to represent aggregate features like "the average distance to an opponent." Defining such mid-level concepts sometimes results in simpler rules.

There are several ILP algorithms for searching the space of possible rules [6]. AI^2 uses the Prolog-based Aleph software package [8], which can conduct both random and heuristic search in the hypothesis space. It selects the rule it finds with the highest F(β) score (a generalization of the more familiar F(1) metric; we use $\beta^2 = 0.1$).

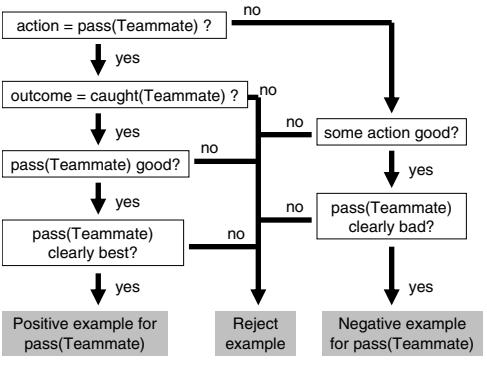

Fig. 3. Example showing how AI^2 selects training examples

To produce datasets for this search, AI^2 examines states from games in the source task and selects positive and negative examples. In a positive example, several conditions must be met: the skill was performed, the desired outcome occurred, the expected Q-value (using the most recent Q-function) is above a minimum score $minQ_{pos}$ and is at least $ratio_{pos}$ times the predicted Q-values of other actions. In a negative example, some other action was performed, the highest Q-value is above a minimum score $minQ'_{neg}$, and the expected Q-value of the skill being learned is at most $ratio_{neg}$ times the highest Q-value in that state and is below a maximum score $maxQ_{neg}$.

The standard settings in AI^2 are $ratio_{pos} = 1.05$ and $ratio_{neg} = 0.95$, since we have found that Q-values in stochastic domains like RoboCup are often not widely separated. The other parameters are set by the system so that there are at least 100 positive and 100 negative examples, which we have found to be enough to learn reasonable rules. Figure 3 illustrates the sorting process with an example from RoboCup.

3.2 Mapping Skills

To produce advice for the new task, the system translates source-task objects into target-task objects based on the user-provided mapping. For example, a reasonable mapping from 4-on-3 KeepAway to 3-on-2 BreakAway might relate each keeper to an attacker and each taker to the defender.

Not all the objects in the target task need to appear in the mapping. Objects in the source task may be left out too; in this case AI^2 will simply not include those objects in rules. Examples in the mapping above are the BreakAway *goalie* and the KeepAway *fieldCenter*. Similarly, the ILP algorithm will leave out of the search space any predicates in the domain that are not shared by both tasks.

The KBKR advice-taking algorithm ultimately requires advice that is propositionalized for a specific task. Therefore, the final step AI^2 takes in the mapping process is to propositionalize the rules.

First it instantiates skills like *pass(Teammate)* for the target task. For 3-on-2 BreakAway, this would produce two rules, *pass(a1)* and *pass(a2)*. Next it deals with any other conditions in the rule body that contain variables. For example, a rule might have this condition:

$$10 < \text{distBetween(a0, Attacker)} < 20$$

This is effectively a disjunction of conditions: either the distance to *a1* or the distance to *a2* is in the interval $[10, 20]$. Since disjunctions are not part of the advice language, AI^2 uses tile features to represent them. Recall that each feature range is divided into Boolean tiles that take value 1 when the feature value falls into their interval and 0 otherwise. This disjunction is satisfied if at least one of several tiles is active; e.g. for 3-on-2 BreakAway:

$$\text{distBetween(a0, a1)}_{[10,20]} + \text{distBetween(a0, a2)}_{[10,20]} \geq 1$$

If these exact tile boundaries do not exist in the target task, AI^2 adds new tile boundaries to the feature space. Thus transfer advice can be expressed exactly even though the target task feature space is unknown at the time the source task is learned.

It is possible for multiple conditions in a rule to refer to the same variable. For example:

distBetween(a0, Attacker) > 15,
angleDefinedBy(Attacker, a0, ClosestDefender) > 25

Here the variable *Attacker* represents the same object in both clauses, so the system cannot propositionalize the two clauses separately. Instead, it defines a new Boolean background-knowledge predicate:

```
newFeature(Attacker, ClosestDefender) :-
   Dist is distBetween(a0, Attacker),
   Ang is angleDefinedBy(Attacker, a0, ClosestDefender),
   Dist > 15, Ang > 25.
```

It then expresses the required condition using the new feature; e.g. for 3-on-2 BreakAway:

$$\text{newFeature(a1, d0)} + \text{newFeature(a2, d0)} \geq 1$$

AI^2 adds these new Boolean features, which could be considered multi-dimensional tiles, to the target task. Thus transfer advice can actually enhance the feature space of the target task.

3.3 User Advice

Users can optionally include advice in the source-target mapping to further guide transfer by pointing out the differences between the tasks. For example, the passing skills transferred from KeepAway to BreakAway make no distinction between passing toward the goal and away from the goal. Since the new objective is to score goals, players should clearly prefer passing toward the goal. A user could provide this guidance by instructing the system to add a condition like this to the *pass(Teammate)* skill:

$$\text{distBetween(a0, goal) - distBetween(Teammate, goal)} \geq 1$$

Alternatively, an expert user could make use of the system's ability to define new features in the target task. The advantage of this approach is that formally defining the feature allows it to be tiled. To do this, the user would first write the definition in Prolog:

```
diffGoalDistance(Teammate, Value) :-
    DistTeammate is distBetween(Teammate, goal),
    DistA0 is distBetween(a0, goal),
    Value is DistA0 - DistTeammate.
```

Then the user would instruct the system to add to the *pass(Teammate)* rule:

$$\text{diffGoalDistance(Teammate)} \geq 1$$

User advice may also describe new skills that will be needed in the target task. An example is the *shoot* skill in BreakAway, which is an important difference from the KeepAway source task. This type of user advice is not required in AI^2, but it provides a natural and powerful way for users to facilitate transfer.

4 Advice Implementation

The rules produced by transfer or provided by users are likely to be imperfect and may even be incorrect. Therefore, obeying them exactly could prevent effective learning. AI^2 instead treats advice as a soft constraint: an RL agent can selectively refine or ignore advice if it disagrees with the agent's experience in the target task.

AI^2 incorporates advice into Q-learning using a linear optimization method called KBKR. The linear optimizer creates a Q-function by finding, for each action, a weight for each state feature so that the Q-value of each state-action pair in the training set is approximately the weighted sum of the features. It does so by minimizing the following quantity:

$$\text{ModelSize} + C \times \text{DataMisfit} + \mu \times \text{AdviceMisfit}$$

Here *ModelSize* is the sum of the absolute values of the feature weights, *DataMisfit* is the disagreement between the learned function's outputs and the training examples, and *AdviceMisfit* is the disagreement between the learned function's outputs and the advice constraints. The numeric parameters C and μ specify the relative importance of minimizing disagreements versus finding a simple model. AI^2 decays μ over time, so that advice fades as the learner gains experience and no longer requires guidance. When μ becomes essentially zero, AI^2 stops applying advice altogether.

As training progresses, this linear program is resolved after every 25 games. See Maclin et al. [5] for more details.

5 Empirical Results

We present results for AI^2 skill transfer between several RoboCup games. These are challenging transfer scenarios because the games have very different reward structures, as described in Section 2. We also include a study of how the KBKR advice-taking algorithm handles imperfect and incorrect advice. Our player code for these experiments is based on the University of Amsterdam Trilearn players.

5.1 Skill Transfer Experiments

In our first experiment, we use AI^2 to perform transfer from 4-on-3 Mobile Keepaway to 3-on-2 BreakAway. The skill we transfer is *pass(Teammate)*, and we use the mapping described in Section 3.2. We assume the user encourages passing toward the goal by adding the *diffGoalDistance* condition from Section 3.3, and approximates some new skills in BreakAway as follows:

```
IF    distBetween(a0, goalLeft) < 10                AND
      angleDefinedBy(goalLeft, a0, goalie) > 40
THEN prefer shoot(goalLeft) over all actions

IF    distBetween(a0, goalRight) < 10               AND
      angleDefinedBy(goalRight, a0, goalie) > 40
THEN prefer shoot(goalRight) over all actions

IF    distBetween(a0, goalCenter) > 10
THEN prefer moveAhead over moveAway and the 3 shoot actions
```

In our second experiment, we perform transfer from 3-on-2 MoveDownfield to 3-on-2 BreakAway using a similar mapping. The skills we transfer are *pass (Teammate)* and *moveAhead*, and we assume the user advice includes only the *shoot* skills above. That is, we assume that passing forward and moving ahead are learned in MoveDownfield, so the user does not need to provide this guidance.

We now analyze the results for the first experiment in detail. AI^2 learned the following rule from Mobile KeepAway:

```
pass(Teammate) :-
   distBetween(k0, Teammate) > 14,
   angleDefinedBy(Teammate, k0, ClosestTaker) ∈ [30, 150],
   distBetween(k0, Taker) < 7,
   distBetween(k0, Player) < 11.
```

This rule indicates that it is good to pass when an opponent is too close, a teammate is somewhat far away, and no opponent is blocking a pass. AI^2 translates this rule into two items of transfer advice, one per BreakAway teammate, and adds in the user advice.

Figure 4 compares learning curves in BreakAway with and without AI^2 transfer from Mobile KeepAway. It also shows learning curves with the transferred skills and the user advice separately, so that we can analyze their individual

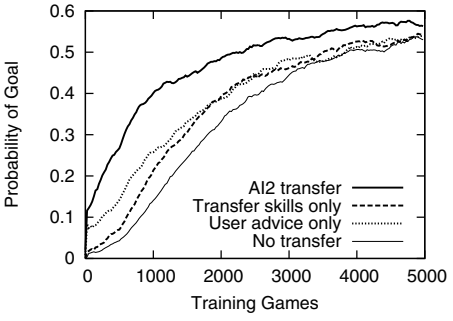

Fig. 4. Learning curves for BreakAway with transfer from Mobile KeepAway

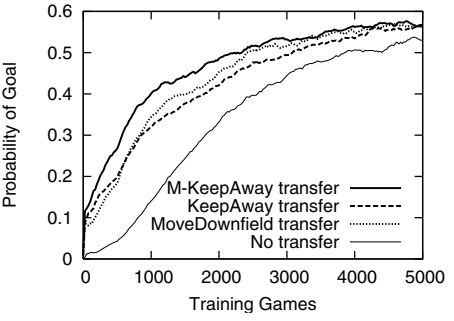

Fig. 5. Learning curves for BreakAway with transfer from several tasks

contributions. Each curve is an average of 10 independent runs with $C = 1500$ and $mu = 10$, and each data point is smoothed over the last 500 games (or all previous games if there are fewer than 500).

Using the transferred skills alone, the scoring probability is higher at the 90% confidence level, based on unpaired t-tests, up to 2500 games. With the full AI^2 system, scoring is more probable at the 95% confidence level at nearly every point. The full system also performs significantly better than either the transferred skills or user advice alone at the 95% confidence level; transferred skills and user hints together perform better than the sum of their parts.

For the second experiment, Figure 5 compares learning curves in BreakAway with and without AI^2 transfer from MoveDownfield. The AI^2 transfer curve for Mobile KeepAway is duplicated here, and we also include results for transfer from the original 3-on-2 KeepAway task [9] to compare the performance of AI^2 transfer in the two KeepAway variants. As expected, Mobile KeepAway transfers better than non-mobile KeepAway. However, both variants as well as MoveDownfield do successfully transfer to BreakAway using AI^2, causing a higher scoring probability at the 95% confidence level at nearly every point.

5.2 Experiments with Imperfect Advice

To demonstrate that AI^2 can cope with imperfect and incorrect advice, we include a third experiment: transfer with intentionally bad user advice. We perform transfer from Mobile KeepAway to BreakAway with the opposite of the user advice above. With its inequalities reversed, this bad advice instructs the learner to pass backwards, shoot when far away from the goal and at a narrow angle, and move when close to the goal.

Figure 6 shows the results of this advice, both in AI^2 transfer and alone. This experiment shows that while bad advice can decrease the positive effect of transfer, it does not cause the AI^2 system to impact learning negatively. On its own, bad advice does have an initial negative effect, but KBKR quickly learns to ignore the advice and the learning curve recovers completely.

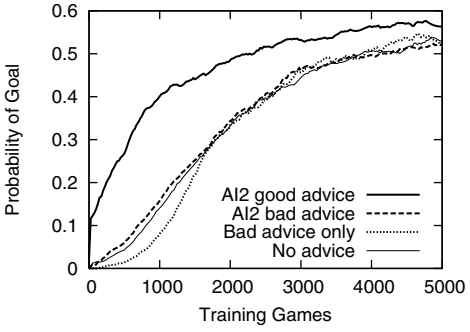

Fig. 6. Learning curve for BreakAway with bad user advice

6 Related Work

Our approach builds on several previous methods for providing advice to reinforcement learners. Maclin and Shavlik [4] develop an IF-THEN advice language to incorporate rules into a neural network for later adjustment. Driessens and Dzeroski [1] use human guidance to create a partial initial Q-function for a relational RL system. Kuhlmann et al. [3] propose a rule-based advice system that increases Q-values by a fixed amount.

Another aspect of our work is extracting explanatory rules from complex functions. Sun [10] studies rule learning from neural-network based reinforcement learners. Fung et al. [2] investigate extracting rules from support vector machines.

We also address knowledge transfer in RL. Singh [7] studies transfer of knowledge between sequential decision tasks. Taylor and Stone [14] copy initial Q-functions to transfer between KeepAway games of different sizes. In Torrey et al. [15] we introduce transfer from KeepAway to BreakAway using the Q-function; we advise each action when its Q-value under the mapped model is highest. In contrast, in this work we learn advice rules in first-order logic to transfer individual skills, working with KeepAway game traces rather than Q-functions. By doing so, we achieve better transfer.

A more detailed study of transfer learning in RoboCup is available online as UW Machine Learning Group Working Paper #06-2.

7 Conclusions and Future Work

Reinforcement learners can benefit significantly from the user-guided transfer of skills from a previous task. We have presented the AI^2 system, which transfers shared skills by learning first-order rules from agent behavior and translating them with a user-designed mapping. This system does not assume a similar reward structure between the source and target tasks and provides robustness to imperfect transfer through advice-taking. Our experimental results demonstrate the effectiveness of this approach in a complex RL domain.

A challenge that we have encountered in RL transfer learning is that differences in action sets and reward structures between the source and target task make it difficult to transfer even shared actions. Changing the game objective or adding a new action changes the meaning of a shared skill. We have addressed this problem with user guidance, using human domain knowledge to help apply transferred skills and encourage the learning of new skills. In the future we hope to reach similar levels of transfer with less user guidance.

We believe that the underlying issue is the separation of *general* from *specific* information in a source task. In RL transfer learning we want to transfer only general aspects of skills in a domain, filtering out task-specific aspects. Our use of ILP to learn general, first-order skill concepts is a step toward this goal. A future step we are considering is learning skills from multiple games in a domain, which we believe may lead to more general rules and therefore better transfer.

References

1. K. Driessens and S. Dzeroski. Integrating experimentation and guidance in relational reinforcement learning. In *Proc. ICML*, 2002.
2. G. Fung, S. Sandilya, and B. Rao. Rule extraction from linear support vector machines. In *Proc. KDD*, 2005.
3. G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, 2004.
4. R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.
5. R. Maclin, J. Shavlik, L. Torrey, T. Walker, and E. Wild. Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In *Proc. AAAI*, 2005.
6. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming 19,20*, pages 629–679, 1994.
7. S. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning 8 (3-4)*, pages 323–339, 1992.
8. A. Srinivasan. The Aleph manual. `http://web.comlab.ox.ac.uk/oucl/research/areas /machlearn/Aleph/aleph.html` , 2001.
9. P. Stone and R. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proc. ICML*, 2001.
10. R. Sun. Knowledge extraction from reinforcement learning. *New Learning Paradigms in Soft Computing*, pages 170–180, 2002.
11. R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning 3*, pages 9–44, 1988.
12. R. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proc. NIPS*, 1996.
13. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
14. M. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *Proc. AAMAS*, 2005.
15. L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proc. ECML*, 2005.