Advice-based Transfer in Reinforcement Learning

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

[1] Computer Science Department, University of Wisconsin, Madison WI 53705, USA
[2] Computer Science Department, University of Minnesota, Duluth, MN 55812, USA

Abstract. This report is an overview of our work on transfer in reinforcement learning using advice-taking mechanisms. The goal in transfer learning is to speed up learning in a target task by transferring knowledge from a related, previously learned source task. Our methods are designed to do so robustly, so that positive transfer will speed up learning but negative transfer will not slow it down. They are also designed to allow human teachers to provide simple guidance that increases the benefit of transfered knowledge. These methods allow us to push the boundaries of current work in this area and perform transfer between complex and dissimilar tasks in the challenging RoboCup simulated soccer domain.

1 Introduction

Reinforcement learning (RL) tasks are often addressed independently, under the implicit assumption that each new task has no relation to the tasks that came before. However, many RL domains contain several related tasks. Instead of learning a new task from scratch, agents in such domains should be able to use knowledge learned in previous tasks to speed up learning in the new task. This is the goal of *transfer learning* in RL.

For example, consider the domain of simulated soccer (e.g., RoboCup [27]). Suppose an RL agent has learned a game of keeping the ball from its opponents by passing among its teammates. Suppose the next game to learn is to score goals against opponents. Since these games have some similarities, the agent could benefit from using its knowledge from the first game while learning the second. In this case we refer to the first game as the *source task* and the second game as the *target task*.

Transfer in RL can be challenging for several reasons. The source and target tasks may differ in their features, actions, or rewards. This means that not all of the knowledge will be transferable (e.g., some actions in the source may not exist in the target) and even transferable knowledge may not be helpful (e.g., an action exists in both, but leads to rewards in the source and not in the target). This uncertainty is compounded by the fact that RL agents may use actions in unintended ways (e.g., using a 'shoot' action to pass to a teammate near the goal, not as an attempt to score a goal).

One straightforward transfer method would be to use the final solution to the source task as the initial solution for the target task, and then perform standard

reinforcement learning to build on that solution incrementally. We refer to this approach as *model reuse*. This method might speed up learning considerably if the task solutions are very similar, but it seems likely to slow down learning if they are even moderately different. Since the agent does not know how similar the solutions will be before actually learning the target task, we would prefer a more robust transfer method. We use the *advice taking* paradigm [20] to achieve this goal.

Advice is a set of instructions about the task solution. For example, here is some advice about passing in soccer:

IF an opponent is near me
AND my nearest teammate is open
THEN pass to nearest teammate

In advice-taking RL algorithms, advice can be followed, refined, or ignored according to its value. We can therefore express source task knowledge as advice – it is followed if it leads to positive transfer, but is quickly refined or ignored if it leads to negative transfer.

When advice comes from a source task, we refer to it as *transfer advice*. Our transfer methods are essentially different ways of getting transfer advice from the source task. Advice can also express direct human guidance, in which case we refer to it as *user advice*. Some of our transfer methods allow user advice to further guide the transfer process, which provides a natural and powerful way for users to interact with the learning system.

2 Background

In this section we provide some background information on reinforcement learning and advice taking. We rely on this material when we discuss our transfer methods later. Furthermore, we describe a motivating domain for transfer in RL: RoboCup simulated soccer. The experiments we present later use this domain.

2.1 Reinforcement Learning and Advice

In reinforcement learning [38], an agent navigates through an environment trying to earn rewards. The environment's state is typically described by a set of features. After each action the agent takes, it receives a reward and observes the next state.

In Q-learning [45], one common form of RL, the agent builds a Q-function to estimate the long-term 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 a reward, the agent updates its Q-value estimates for the current state.

In our RL implementation, agents build Q-functions using the $SARSA$ and TD(λ) procedures [37]. We use $\lambda = \exp(-age/100)$ where the age of a game

is the number of games the learner trained on before that game; older games therefore have smaller λ. Our learning rate has an initial value $\alpha = 0.5$ and a half-life of 1000 games. The form of the Q-function for each action is a weighted linear sum of the state features. Rather than updating their Q-functions after every action, our agents play batches of 25 full games at a time. After each batch they solve a linear optimization problem to find the weights that minimize:

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

Here *ModelSize* is the sum of the absolute values of the feature weights, and *DataMisfit* is the disagreement between the learned function's outputs and the training examples. The numeric parameter C specifies the relative importance of minimizing disagreement with the data versus finding a simple model.

We incorporate advice into this RL algorithm by adding another term to the optimization problem, so that it minimizes:

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

Here *AdviceMisfit* is the disagreement between the learned function's outputs and the advice constraints. The numeric parameter *mu* specifies the relative importance of minimizing disagreement with the advice versus minimizing the original quantity. Over time *mu* decays so that advice fades as the learner gains experience and no longer requires guidance, and C correspondingly increases.

Advice is therefore a set of soft constraints on the task solution. Depending on whether it agrees with the training examples, the learner can follow the advice, only follow it approximately (which is like refining it), or ignore it altogether. This RL algorithm is based on *Knowledge-Based Kernel Regression* [20, 21, 24]. The next section goes into further technical detail about KBKR.

There is a substantial body of related work in advice taking. One approach that is quite different from ours, but might also be considered advice taking, is *expert imitation*. Examples of this are Lin [17], who replays teacher sequences to bias a learner towards a teacher's performance, and Sammut et al. [30], who use imitation of human experts to train a flight simulation program.

Another form of advice puts direct constraints on the policies that RL agents learn. Andre and Russell [1] describe a language for giving policy constraints to learning agents. Kuhlmann et al. [16] propose a rule-based advice system that increases Q-values by a fixed amount. Our method differs from these in that the user only needs to specify action preferences, not internal details like desired Q-values.

Some advice is only used at the beginning of the learning process, as in Driessens and Dzeroski [6], who use human guidance to create a partial initial Q-function in relational RL. Other advice is used only at a specific step, as in Clouse and Utgoff [4], who allow a human observer to step in and advise the learner to take a specific action. Our method differs from these in that the advice is present throughout the learning process, although its impact decreases over time.

There are several methods that use advice rules that look like ours. Gordon and Subramanian [13] accept advice in the form IF *condition* THEN *achieve goals* and use genetic algorithms to adjust it with respect to the data. Maclin and Shavlik [19] develop an IF-THEN advice language to incorporate rules into a neural network for later adjustment. Our method differs from these because we incorporate the rules into an optimization problem instead of using neural networks or genetic algorithms.

2.2 Knowledge-Based Kernel Regression

In this section we explain the KBKR advice-taking algorithm in more detail and note the settings that we used for our transfer experiments. This information is technical and need not be read in order to understand the transfer learning concepts, but it explains more precisely how advice is applied in our system.

As we mentioned above, after each batch of games the RL agent in KBKR uses its training examples to build a Q-function model. The form of the Q-function for each action is a weighted linear sum of the state features, which means the agent is actually finding an optimal weight vector w that has one weight for each feature in the feature vector x. Each action has its own weight vector and offset term b, and the expected Q-value of taking that action from the state described by x is $wx + b$. Our learners take the action that scores the highest with probability $(1 - \epsilon)$, and take a sub-optimal exploratory action with probability ϵ. In our transfer experiments we used an initial $\epsilon = 0.025$ and decayed it exponentially with a half-life of 5000 episodes.

To compute the weight vector for an action, we find the subset of training examples in which that action was taken and place those feature vectors into rows of a data matrix A of size 1000. When there are too many examples to fit into A, we begin to discard games randomly such that the probability of discarding a game increases exponentially with the age of the game. Using the current model and the actual rewards received in the examples, we compute Q-value estimates and place them into an output vector y. The optimal weight vector is then described by

$$Aw + b\overrightarrow{e} = y \tag{1}$$

where $\overrightarrow{e}$ denotes a vector of ones (we omit this for simplicity from now on).

For our transfer experiments, the matrix A contains 25% exploration examples and 75% regular exploitation examples. We do this so that bad moves are not forgotten, as they would be if we used almost entirely exploitation examples. Since the exploration rate is only 2.5%, we create enough exploration examples by randomly choosing exploitation steps and using the model to score off-policy actions for those steps.

In practice, we prefer to have non-zero weights for only a few important features in order to keep the model simple and avoid overfitting the training examples. We therefore introduce *slack* variables s that allow inaccuracies on

some examples, and a penalty parameter C for trading off these inaccuracies with the complexity of the solution. The resulting minimization problem is

$$\begin{array}{ll} \min\limits_{(w,b,s)} & ||w||_1 + \nu|b| + C||s||_1 \\ s.t. & -s \leq Aw + b - y \leq s. \end{array} \tag{2}$$

where $|\cdot|$ denotes an absolute value, $||\cdot||_1$ denotes a sum of absolute values, and ν is a penalty on the offset term. By solving this problem, we can produce a weight vector w for each action that compromises between accuracy and simplicity. For our experiments, we used $\nu = 100$ and $C = 1000$ decaying exponentially with a half-life of 2500 episodes.

In Mangasarian et al.'s [24] original formulation, called Knowledge Based Kernel Regression (KBKR), advice can be given in the form of a rule about a single action. This rule creates new constraints on the problem solution, in addition to the constraints from the training data. More recently, we introduced an extension to KBKR called Preference-KBKR [21], which allows advice about pairs of actions in the form

$$Bx \leq d \Longrightarrow Q_p(x) - Q_n(x) \geq \beta, \tag{3}$$

which can be read as:

> If the current state satisfies $Bx \leq d$, the Q-value of the preferred action p should exceed that of the non-preferred action n by at least β.

For example, consider giving the advice that shooting is better than moving ahead when the distance to the goal is at most 10. The vector B would have one row with a 1 in the column for the "distance to goal" feature and zeros elsewhere. The vector d would contain only the value 10, and β could be set to some small positive number.

Just as we allowed some inaccuracy on the training examples, we allow advice to be followed only partially. To do so, we introduce slack variables z and ζ and penalty parameters μ_1 and μ_2 for trading off the impact of the advice on the solution with the impact of the training examples. For our experiments, we used $\mu_1 = 1$ and $\mu_2 = 100$, both decaying exponentially with a half-life of 1000 episodes.

The new minimization problem addresses all the actions together so that it can apply constraints to their relative values. Multiple pieces of preference advice can be incorporated, each with its own B, d, p, n, and β. This makes it possible to advise taking a particular action (by stating that it is preferred over all the other actions). We use the CPLEX commercial software program to solve the resulting linear program:

$$\min_{(w_a, b_a, s_a, z_i, \zeta_i \geq 0, u_i \geq 0)} \sum_{a=1}^{m} (||w_a||_1 + \nu |b_a| + C||s_a||_1) + \sum_{i=1}^{k} (\mu_1 ||z_i||_1 + \mu_2 \zeta_i) \quad (4)$$

$$\begin{aligned} \text{s.t.} \quad & \text{for each action } a \in \{1, \ldots, m\} : \\ & -s_a \leq A_a w_a + b_a - y_a \leq s_a \\ & \text{for each piece of advice } i \in \{1, \ldots, k\} : \\ & -z_i \leq w_p - w_n + B_i^T u_i \leq z_i \\ & -d^T u_i + \zeta_i \geq \beta_i - b_p + b_n. \end{aligned}$$

Note that while this method is called Kernel Regression, we do not actually use a kernel, because we found that the non-kernelized version worked best.

We have also developed a version of KBKR called ExtenKBKR [22], which incorporates advice in another way designed to allow a higher volume of advice. Some of our transfer experiments require this version, since they produce large amounts of advice. We refer the reader to Maclin et al. [22] for details on that method.

2.3 RoboCup

One motivating domain for transfer in RL is *RoboCup simulated soccer*. The RoboCup project [27] has the overall goal of producing robotic soccer teams that compete on the human level, but it also has a software simulator for research purposes. Stone and Sutton [35] introduced RoboCup as an RL domain that is challenging because of its large, continuous state space and nondeterministic action effects.

Since the full game of soccer is quite complex, researchers have developed several smaller games in the RoboCup domain (see Figure 1). These are inherently multi-agent games, but a standard simplification is to have only one agent (the one in possession of the soccer ball) learning at a time using a model built with combined data from all the agents.

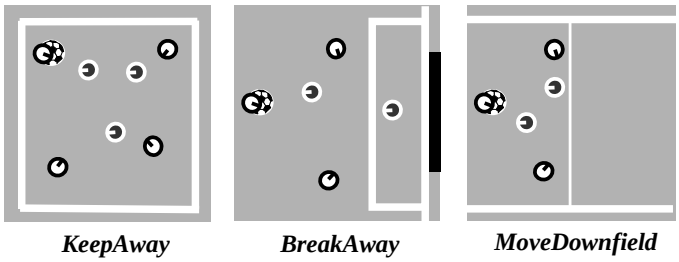

Fig. 1. Snapshots of RoboCup soccer tasks.

The first RoboCup task we use is M-on-N KeepAway [35], in which the objective of the M reinforcement learners called *keepers* is to keep the ball away from N hand-coded players called *takers*. The keeper with the ball may choose either to hold it or to pass it to a teammate. Keepers without the ball follow a hand-coded strategy to receive passes. 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.

Our KeepAway state representation is based on the one designed by Stone and Sutton [35]. The features are listed in Table 1. The keepers are ordered by their distance to the learner *k0*, as are the takers.

Note that we present these features as predicates in first-order logic. Variables are capitalized and typed (*Player*, *Keeper*, etc.) and constants are uncapitalized. For simplicity we indicate types by variable names, leaving out implied terms like *player(Player)*, *keeper(Keeper)*, etc. Since we are not using fully relational reinforcement learning, the predicates are actually grounded and used as propositional features during learning (and for the first two transfer methods). However, our final transfer method uses the features in their first-order representation, so we show that version here.

A second RoboCup task is M-on-N MoveDownfield, where the objective of the M reinforcement learners called *attackers* is to move across a line on the opposing team's side of the field while maintaining possession of the ball. The attacker with the ball may choose to pass to a teammate or to move ahead, away, left, or right with respect to the opponent's goal. Attackers without the ball follow a hand-coded strategy to receive passes. The game ends when they cross the line, when an opponent takes the ball, when the ball goes out of bounds, or after a time limit of 25 seconds. The learners receive symmetrical positive and negative rewards for horizontal movement forward and backward.

Our MoveDownfield state representation is the one presented in Torrey et al. [41]. The features are listed in Table 1. The attackers are ordered by their distance to the learner *a0*, as are the defenders.

A third RoboCup task is M-on-N BreakAway, where the objective of the M attackers is to score a goal against $N - 1$ hand-coded *defenders* and a hand-coded *goalie*. The attacker with the ball may choose to pass to a teammate, to move ahead, away, left, or right with respect to the opponent's goal, or to shoot at the left, right, or center part of the goal. Attackers without the ball follow a hand-coded strategy to receive passes. The game ends when they score a goal, 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.

Our BreakAway state representation is the one presented in Torrey et al. [43]. The features are listed in Table 1. The attackers are ordered by their distance to the learner *a0*, as are the non-goalie defenders.

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

Table 1. RoboCup task feature spaces.

KeepAway features
distBetween(k0, Player) distBetween(Keeper, minDistTaker(Keeper)) angleDefinedBy(Keeper, k0, minAngleTaker(Keeper)) distBetween(Player, fieldCenter)
MoveDownfield features
distBetween(a0, Player) distBetween(Attacker, minDistDefender(Attacker)) angleDefinedBy(Attacker, a0, minAngleDefender(Attacker)) distToRightEdge(Attacker) timeLeft
BreakAway features
distBetween(a0, Player) distBetween(Attacker, minDistDefender(Attacker)) angleDefinedBy(Attacker, a0, minAngleDefender(Attacker)) distBetween(Attacker, goalPart) distBetween(Attacker, goalie) angleDefinedBy(Attacker, a0, goalie) angleDefinedBy(GoalPart, a0, goalie) angleDefinedBy(topRightCorner, goalCenter, a0) timeLeft

20 units away from *a0* and 0 otherwise. Stone and Sutton [35] found tiling to be important for timely learning in RoboCup. It also gives our linear Q-function model the ability to represent complex nonlinear functions.

These three RoboCup games have substantial differences in features, actions, and rewards. The goal, goalie, and shoot actions exist in BreakAway but not in the other two tasks. The move actions do not exist in KeepAway but do in the other two tasks. Rewards in KeepAway and MoveDownfield occur for incremental progress, but in BreakAway the reward is more sparse. These differences mean the solutions to the tasks may be quite different. However, some knowledge should clearly be transferable between them, since they share many features and some actions, such as the *pass* action. Furthermore, since these are difficult RL tasks, speeding up learning through transfer would be desirable.

3 Transfer in Reinforcement Learning

The goal in transfer learning is to speed up learning in a *target* task by transferring knowledge from a related *source* task. To design a method for transfer in reinforcement learning, we need to answer three questions:

- What knowledge will we transfer from the source task?
- How will we extract that knowledge from the source task?
- How will we apply that knowledge in the target task?

To put our methods in context, we will discuss a range of possible answers to these questions (and mention relevant related work).

3.1 Types of Knowledge in the Source Task

Perhaps the most straightforward type of knowledge learned in the source task is the actual solution to the source task: the Q-functions (or whatever type of model is being learned). Taylor et al. [40] and Taylor and Stone [39] transfer Q-functions, and Gorski and Laird [14] transfer value functions.

Another type of knowledge is the *policy* created by the model; that is, the choice of which action to take in each state. This information consists of action preferences, rather than the explicit values of actions. We transfer policies in Torrey et al. [43], which we discuss further in Section 5.

A *skill* is like one piece of a policy: the circumstances in which to take one specific action. Skills can express more general information than a full policy. We transfer skills in Torrey et al.[41, 42], which we discuss further in Section 6.

In domains where there are many possible actions, so that considering every action is too expensive, the source task may provide information on which actions are important and which can be ignored. Sherstov and Stone [31] transfer this type of knowledge.

Multi-step actions are known as *options*, and the source task may provide insight into which of the many possible action sequences would make useful

options. Perkins and Precup [28] and Soni and Singh [33] transfer this type of knowledge.

If there is an explicit or implicit Markov Decision Process (MDP) that describes the RL task, then we could transfer information about states and transitions in the MDP. Asadi et al. [2], Ferguson and Mahadevan [10] and Walsh et al. [44] describe approaches in this vein.

3.2 Methods for Extracting Knowledge from the Source Task

The types of knowledge mentioned above could be extracted from the source task in two general ways: by analyzing the underlying model (e.g, the Q-function), or by observing the agent's successful behavior in the source task.

Analyzing the underlying model can be as simple as copying the model directly, as in Taylor et al. [40]. We take a slightly different approach in Torrey et al. [43]: using the source-task model to find the highest-value target-task action.

By observing successful behavior in the source task, learners can imitate expert agents – either human (as in Sammut et al. [30]) or electronic (as in Price and Boutilier [29]). We also use observed games in Torrey et al. [41] by learning rules that described skills.

3.3 Methods for Applying Knowledge in the Target Task

In almost any method of applying source-task knowledge in a target task, a *mapping* between the two tasks is required. The mapping shows how the tasks are related by matching up corresponding features and actions. Standard practice is currently to assume a human provides this information, although some recent work by Liu and Stone [18] uses analogical structure mapping [9] to acquire it automatically.

One alternative that should be mentioned here is relational reinforcement learning (RRL), which can make mapping unnecessary. If the domain can be formulated using RRL in a general way that applies to both the source and target task, then no translation between them may be required. Driessens et al. [7] and Stracuzzi and Asgharbeygi [36] describe ways to apply RRL to transfer learning.

However, assuming a mapping exists to translate source-task knowledge into target-task terms, there are several possible ways to apply the knowledge. One that we have mentioned already is *model reuse*: using the final model for the source task as the initial model for the target task, and performing normal RL from there. Singh [32], Mehta et al. [25], and Taylor et al. [40] are examples of this approach.

Another method is to follow source task policies during the exploration steps of normal RL in the target task, instead of doing random exploration. This approach is sometimes referred to as *policy reuse*. Madden and Howley [23] and Fernandez and Veloso [11, 12] are examples of this approach.

One technique sometimes used to speed up normal RL is *reward shaping*, in which the designer of an RL task deliberately constructs its rewards in a way

that helps the learner toward a solution. Konidaris and Barto [15] apply this idea to transfer by using source-task knowledge to shape rewards.

The approach that we focus on is *advice taking*. As we have explained, advice can be viewed as a set of soft constraints on the task solution. Our transfer methods in Torrey et al. [43] and Torrey et al. [41] construct advice from the source task and then use an advice-taking RL algorithm to learn the target task.

3.4 Our Transfer Methods

Our work so far has produced two advice-based transfer methods, which we have mentioned above. The first is called *policy transfer*, in which we encourage the RL agents to apply the source-task policy (as specified by its Q-functions) in the target task. The second is called *skill transfer*, in which we give advice about skills learned by observation of the source task.

In the following sections, we provide results from experiments with these transfer methods. We also include experiments on *model reuse* as a baseline approach.

4 Model Reuse

Model reuse is our baseline transfer method of using the final source-task model as the initial target-task model. In our experiments, the model is a Q-function that gives the value of each action as a weighted sum of the features. We assume that the user provides a mapping that shows how (propositional) features and actions correspond between the tasks. Using this mapping, we translate the source-task model into a model for the target task.

The actions in the two tasks need not have one-to-one correspondences. If an action in the source does not exist in the target, we simply ignore the Q-function for that action. For example, we ignore the *hold* action when we transfer from KeepAway to BreakAway. If an action exists in the target that did not exist in the source, we initialize the Q-function for that action with all zero weights. This applies to the *shoot* actions when we transfer from KeepAway to BreakAway. We also allow a source-task action to be mapped to more than one action in the target task; each such mapping can have a separate set of feature mappings. This allows, for example, the single *pass* action in 2-on-1 BreakAway to be mapped to both of the *pass* actions in 3-on-2 BreakAway. In one mapping, the player *a1* in the source is mapped to *a1* in the target, and in the second one, it is mapped to *a2* in the target.

The features in the two tasks might also be only partially overlapping. If a feature in the source does not exist in the target, it needs to be mapped to a numeric value that is typical or average for that feature. We then replace the feature with that constant in the translated Q-function. This applies to the *distanceToCenter* feature when we transfer from KeepAway to BreakAway. If a feature exists in the target that did not exist in the source, we simply give it a zero weight in the translated Q-function. When a feature does exist in both

tasks but with different ranges, we allow the mapping to specify that feature values should be scaled or shifted accordingly. Also recall that there are tiles representing segments of each numeric feature; we automatically map each tile in the source task to a tile in the target task (the one for which the ratio of overlap area between the tile intervals to non-overlap area is highest).

Given a translated model, we perform model reuse in our RL system by having the agent play the target task using that model for 10 batches of 25 games. After this, it learns normally by solving the optimization problem described in Section 2.1 after every batch to produce new Q-functions. Note that since our RL system relearns Q-functions completely after each batch instead of changing them incrementally, it is necessary to play with the translated model at first to generate training examples that cause the model's influence to persist after learning.

To help the model persist further, we found it useful to add some additional examples during the first few batches of learning. We create these additional examples in the same way that extra exploration examples are created in KBKR, as described in Section 2.2. We start with 100 of these pseudo-examples when first learning and decay the number linearly down to zero during the second 10 batches of 25 games.

Model Reuse Results

Figures 2, 3, and 4 display target-task learning curves from several transfer experiments using this method. One curve is always the average of 25 runs of standard reinforcement learning. The other curves are RL with transfer via model reuse from various source tasks. For each transfer curve we average 5 transfer runs from 5 different source runs, for a total of 25 runs (this way, the results include both source and target variance). Because the variance is high, we smooth the y-value at each data point by averaging over the y-values of the last 10 data points. The mappings we use for these experiments are listed in the Appendix (Section A.1).

For each task, we use an appropriate measure of performance to plot against the number of training games. In BreakAway, it is the probability that the agents will score a goal in a game. In MoveDownfield, it is the average total reward acquired in a game. In KeepAway, it is the average length of a game.

Each figure has one curve for which the source task is the same as the target task, except with each team size decreased by one. We refer to this as *close transfer* since the tasks are closely related. There are close transfer results for all three subtasks. The remaining curves in Figure 2 have different source and target tasks but equivalent team sizes. We refer to this as *distant transfer* since the tasks are more distantly related. With distant transfer we concentrate on transfer from easier tasks to harder tasks (from KeepAway to BreakAway and from MoveDownfield to BreakAway).

For distant transfer, model reuse produces an immediate performance decrease that takes half the learning curve to recover. This is what we expected: model reuse produces negative transfer when tasks are not closely related.

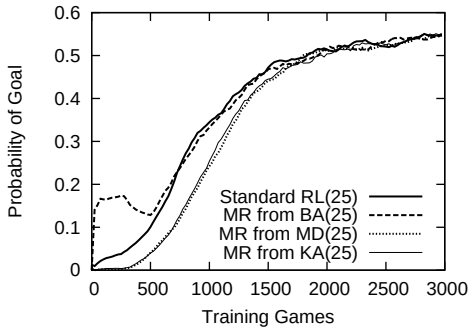

Fig. 2. Probability of scoring a goal while training in 3-on-2 BreakAway with standard RL and model reuse (MR) from 2-on-1 BreakAway (BA), 3-on-2 MoveDownfield (MD) and 3-on-2 KeepAway (KA).

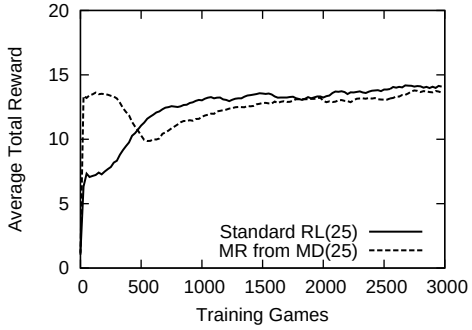

Fig. 3. Average total reward while training in 4-on-3 MoveDownfield with standard RL and model reuse (MR) from 3-on-2 MoveDownfield (MD).

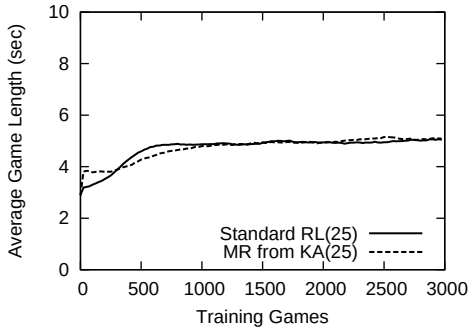

Fig. 4. Average game length while training in 4-on-3 KeepAway with standard RL and model reuse (MR) from 3-on-2 KeepAway (KA).

Table 2. Statistical results from model reuse (MR) experiments in BreakAway (BA), MoveDownfield (MD), and KeepAway (KA), comparing area under the curve to standard reinforcement learning (SRL).

Experiment	*Conclusion*	*p-value*	*95% confidence interval*
BA to BA	MR higher with 92% confidence	0.0740	-11.86, 101.15
MD to BA	SRL higher with 99% confidence	0.0005	-169.52, -49.93
KA to BA	SRL higher with 99% confidence	0.0012	-161.25, -40.41
MD to MD	MR and SRL equivalent	0.3796	-871.30, 1227.32
KA to KA	MR and SRL equivalent	0.4776	-199.32, 214.36

For close transfer, model reuse always produces an immediate performance increase over standard RL. However, it quickly falls to or below the level of standard RL once the players begin learning instead of using the old model directly. This probably would not happen in an incremental learning system, as in Taylor et al. [40], because there would be no sudden changes like this. However, in our RL system, model reuse has mixed results even for close transfer.

To compare the learning curves quantitatively, we use a randomization test to calculate a p-value for the areas under the curves. This p-value is the probability that a t-statistic measured on these areas would be as high if the transfer learning curves were not inherently different from the standard RL curves. A low p-value means that we are more confident that the curves with higher area are in fact significantly higher. To estimate the magnitude of this difference, we also calculate a bootstrapped 95% confidence interval. These performance measures are derived from work by Paul Cohen's group for the DARPA Transfer Learning program [5].

Table 2 summarizes these statistical results for our model reuse experiments. For each experiment it gives a p-value, a confidence level for which curves have higher area, and a 95% confidence interval for the average transfer curve area minus the average standard RL curve area.

The statistical results show that model reuse does not reliably increase total area under the learning curve, even though for close transfer the graphs show that it provides an initial benefit. For distant transfer, model reuse significantly decreases the total area. The advice-based transfer methods we present in the next two sections are more robust.

5 Policy Transfer via Advice

Policy transfer via advice taking is a transfer method in which we advise the source-task policy (as specified by its Q-functions) in the target task. Rather than reusing the Q-functions directly, this method transfers the *policy* created by the Q-functions. It advises the target-task learner to take the same actions

Table 3. A simple example of constructing policy-transfer advice. The actions in the old task are a, b, and c, and the corresponding actions in the new task are a', b', and c'. The learned model for the old task is a set of linear Q-value expressions with weights w and features f, and these are translated into advice that uses the corresponding new task features f'.

SOURCE TASK MODEL:	ADVICE FORMAT:
$Q_a = w_{a1} * f_1 + w_{a2} * f_2$	IF $Q'_a - Q'_b \geq \Delta$
$Q_b = w_{b1} * f_1$	THEN prefer a' to b'
$Q_c = w_{c2} * f_2$	(and so on for each pair of actions)
USER-PROVIDED MAPPING:	FULL ADVICE EXPRESSION:
$(a, b, c) \longrightarrow (a', b', c')$	IF $(w_{a1} - w_{b1}) * f'_1 + w_{a2} * f'_2 \geq \Delta$
$(f_1, f_2) \longrightarrow (f'_1, f'_2)$	THEN prefer a' to b'
TRANSLATED MODEL:	(and so on for each pair of actions)
$Q'_a = w_{a1} * f'_1 + w_{a2} * f'_2$	
$Q'_b = w_{b1} * f'_1$	
$Q'_c = w_{c2} * f'_2$	

that it would have in the source task, but leaves the learner free to determine the actual Q-values of those actions.

We have already described how we give advice to an RL agent and explained why advice taking is a robust mechanism for transfer. Therefore, to finish describing this method we need only explain how we build the transfer advice.

As with model reuse, we assume that the user provides a mapping of (propositional) features and actions between the tasks. We use the same strategies for dealing with partially overlapping feature and action sets, and we continue to allow shifting, scaling, and multiple mappings.

After applying the mapping, we can take a state from the target task and evaluate the Q-values of actions in that state using the translated model. We can then compare all pairs of target-task actions that have source-task analogues, and advise that the higher-scoring action should be preferred over the lower-scoring action. Table 3 gives a simple but concrete example of this process. We set Δ to approximately 1% of the target-task Q-value range.

Policy Transfer Results

Figures 5, 6, and 7 display results from several policy-transfer experiments. They represent the same close and distant transfer scenarios as in the previous section, performed with the same experimental methodology. We use the same mappings as for the model-reuse experiments, as listed in the Appendix (Section A.1).

These graphs show that unlike model reuse, policy transfer can have a small overall positive impact in both close and distant transfer scenarios. We performed the same statistical analysis described in the previous section, and the results in Table 4 indicate that in most cases the policy transfer curves have significantly higher area than the standard RL curves. While the impact of policy transfer is small, practically speaking, it is more robust than model reuse.

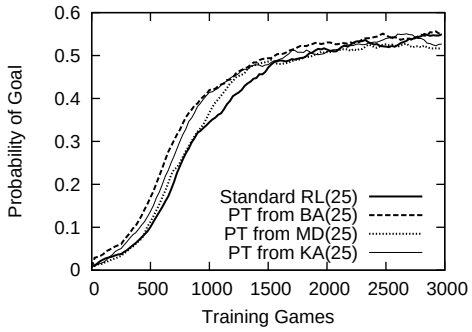

Fig. 5. Probability of scoring a goal while training in 3-on-2 BreakAway with standard RL and policy transfer (PT) from 2-on-1 BreakAway (BA), 3-on-2 MoveDownfield (MD) and 3-on-2 KeepAway (KA).

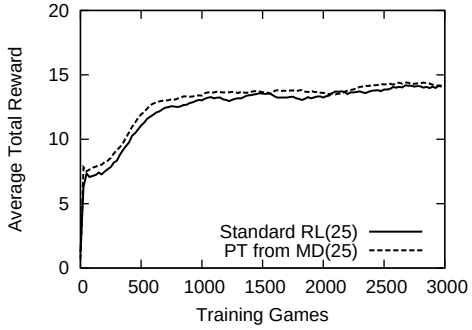

Fig. 6. Average total reward while training in 4-on-3 MoveDownfield with standard RL and policy transfer (PT) from 3-on-2 MoveDownfield (MD).

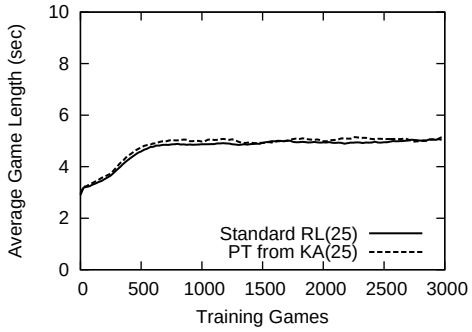

Fig. 7. Average game length while training in 4-on-3 KeepAway with standard RL and policy transfer (PT) from 3-on-2 KeepAway (KA).

Table 4. Statistical results from policy transfer (PT) experiments in BreakAway (BA), MoveDownfield (MD), and KeepAway (KA), comparing area under the curve to standard reinforcement learning (SRL).

Experiment	*Conclusion*	*p-value*	*95% confidence interval*
BA to BA	PT higher with 99% confidence	0.0013	39.49, 160.69
MD to BA	PT and SRL equivalent	0.4015	-53.69, 70.67
KA to BA	PT higher with 97% confidence	0.0301	0.26, 121.04
MD to MD	PT higher with 98% confidence	0.0208	118.04, 2522.17
KA to KA	PT higher with 99% confidence	0.0045	92.78, 510.08

Policy transfer produces a large amount of complex advice – so much that we had to use a variant of KBKR, called ExtenKBKR [22], that handles high advice volumes. Furthermore, like model reuse, this method relies on the low-level, task-specific Q-functions to perform transfer. Our second transfer method in the next section attempts to transfer higher-level knowledge, which leads to larger performance gains.

6 Skill Transfer via Advice

A main objective in transfer learning is to determine which source-task knowledge is *general* (and presumably transferable) and which is *specific* (and presumably non-transferable). The Q-functions combine these two types of knowledge, so methods like model reuse and policy transfer that use Q-functions cannot separate them.

Skill transfer is a method designed to capture general knowledge from the source task and filter out specific knowledge. Instead of transferring an entire policy, this method transfers only the *skills* that the source and target tasks have in common. Furthermore, instead of using Q-functions to describe skills, this method uses inductive logic programming (ILP) [26] to learn skill concepts that generalize over games played in the source task. We learn first-order rules because they can be more general than propositional rules, since they can contain variables. 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.

Figure 8 shows an example of the skill transfer process in the context of transfer from KeepAway to BreakAway. In this example, KeepAway games provide training examples for the concept "states in which passing to a teammate is a good action." An ILP algorithm then learns a rule representing the *pass* skill. Finally, a mapping is applied to produce transfer advice for BreakAway.

Note that we represent states and actions using first-order predicates for this method. During learning in the source and target tasks, both features and actions are still propositional; we only "lift" them into first-order logic temporarily

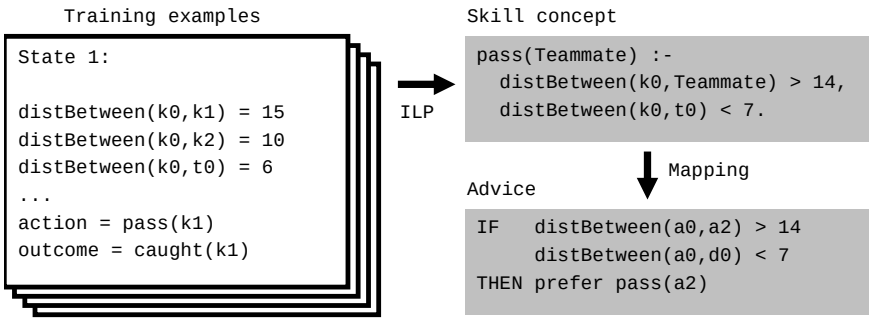

Fig. 8. Example showing how we transfer skills.

during the transfer process. In a first-order representation, matching feature and action predicates are identical throughout the domain, so there is no need to map them. However, we assume the user provides a mapping between logical objects in the source and target tasks (e.g., *k0* in KeepAway maps to *a0* in BreakAway).

The problem of partially overlapping feature sets has a simple solution in this transfer method, because the ILP algorithm for learning advice rules is able to limit its search space to a subset of the feature predicates. We therefore allow only feature predicates that exist in the target task to appear in advice rules. This forces the algorithm to find skill definitions that are relevant in the target task.

6.1 Learning Skills

There are several ILP algorithms for searching the space of possible rules [26]. We use the Prolog-based Aleph software package [34], which can conduct both random and heuristic search in the hypothesis space. The skill transfer method selects the rule it finds with the highest F(β) score (a generalization of the more familiar F(1) metric; we use $\beta^2 = 0.1$ to put more weight on rule precision than rule recall).

To produce datasets for this search, the skill-transfer method examines states from games in the source task and selects positive and negative examples. We found that not all states should be used as training examples; some are not unambiguously positive or negative and should be left out of the datasets. These states can be detected by looking at their Q-values, as described below. Figure 9 summarizes the overall process with an example from RoboCup.

In a good positive example, several conditions should be met: the skill is performed, the desired outcome occurs, the expected Q-value (using the most recent Q-function) is above the 10th percentile in the training set and is at least 1.05 times the predicted Q-values of all other actions. The purpose of these conditions is to remove ambiguous examples in which several actions may be good or no actions seem good.

There are two types of good negative examples. These conditions describe one type: some other action is performed, that action's Q-value is above the

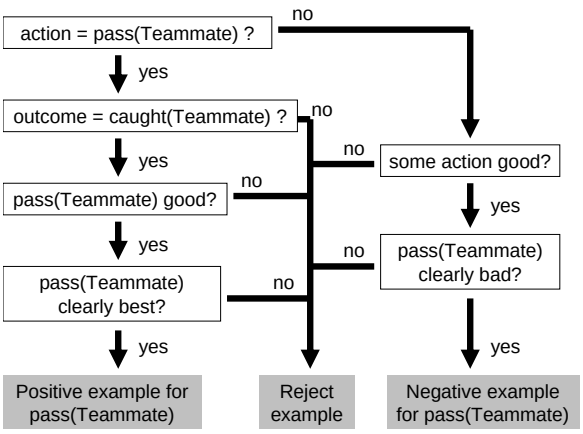

Fig. 9. Example showing how we select training examples.

10th percentile in the training set, and the Q-value of the skill being learned is at most 0.95 times that Q-value and below the 50th percentile in the training set. These conditions remove similarly ambiguous examples. The second type of good negative example includes states in which the skill being learned was taken but the desired outcome did not occur.

To make the search space finite, it is necessary to replace continuous features (like distances and angles) with finite sets of discrete features. For example, the rule in Figure 9 contains the Boolean constraint *distBetween(k0,t0)* < 7, derived from the continuous distance feature. The skill transfer method finds the 25 thresholds with the highest information gain and allows the intervals above and below those thresholds to appear as constraints in rules. Furthermore, we allow up to 7 constraints in each rule. We found these parameters to produce reasonable running times for RoboCup, but they should be adjusted appropriately for other domains.

6.2 Converting Skills to Transfer Advice

To convert a skill concept into transfer advice, we need to apply an object mapping and propositionalize the rule. Propositionalizing is necessary because the KBKR advice-taking algorithm only works with propositional advice. This automated process preserves the meaning of the first-order rules without losing any information, but there are several technical details involved.

First we instantiate skills like *pass(Teammate)* for the target task. For 3-on-2 BreakAway, this would produce two rules, *pass(a1)* and *pass(a2)*. Next we deal 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, we use tile features to represent them. Recall that each feature range is divided into Boolean tiles that take the 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; for 3-on-2 BreakAway this is:

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

If these exact tile boundaries do not exist in the target task, we add 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 that puts simultaneous constraints on both features:

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

It then expresses the entire condition using the new feature; for 3-on-2 BreakAway this is:

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

We add these new Boolean features, which could be considered multi-dimensional tiles, to the target task. Thus skill transfer can actually enhance the feature space of the target task.

Each advice item produced from a skill says to prefer that skill over the other actions shared between the source and target task. As in policy transfer, we set the preference amount Δ to approximately 1% of the target-task Q-value range.

6.3 User Advice

Compared to policy transfer, skill transfer produces a small number of simple, interpretable rules. We believe this introduces the possibility of further user input in the transfer process. If users can understand the transfer advice, they may wish to add to it, either further specializing rules or writing their own rules for new, non-transferred skills in the target task. The skill transfer method therefore allows optional *user advice*.

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:

distBetween(a0, goal) - distBetween(Teammate, goal) ≥ 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:

diffGoalDistance(Teammate, Value),
Value ≥ 1

There are also several actions in this transfer scenario that are new in the target task, such as *shoot* and *moveAhead*. We allow users to write simple rules to approximate skills like these, such as:

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

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

User advice provides a natural and powerful way for users to facilitate transfer beyond providing a mapping.

6.4 Skill Transfer Results

Figures 10, 11, and 12 display results from several skill-transfer experiments. They represent the same close-transfer and distant-transfer scenarios as in the previous two sections, performed with the same experimental methodology. The mappings and user advice that we use for these experiments are listed in the Appendix (Sections A.2 and A.3), as are examples of some learned skill concepts (Section A.4).

These graphs show that skill transfer can have a large overall positive impact in both close-transfer and distant-transfer scenarios. We performed the same statistical analysis as in the previous two sections, and the results in Table 5 indicate that in most cases the skill transfer curves have significantly higher area than the standard RL curves. Furthermore, a comparison of the confidence intervals with those from Table 4 confirms that the impact tends to be larger than with policy transfer.

Because skill transfer has the robustness of policy transfer and can also produce larger performance gains, it is currently our preferred transfer method.

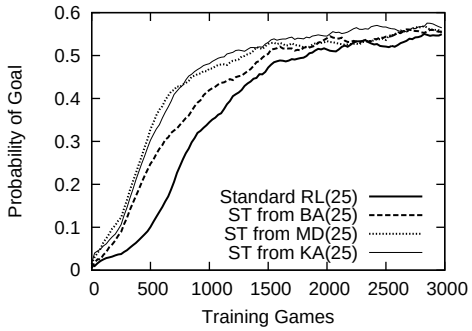

Fig. 10. Probability of scoring a goal while training in 3-on-2 BreakAway with standard RL and skill transfer (ST) from 2-on-1 BreakAway (BA), 3-on-2 MoveDownfield (MD) and 3-on-2 KeepAway (KA).

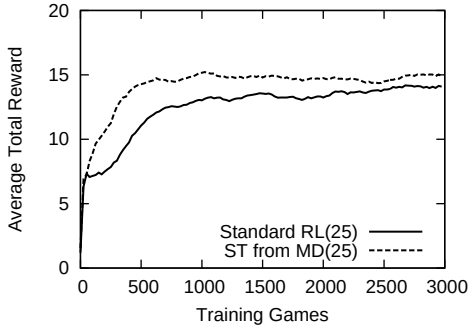

Fig. 11. Average total reward while training in 4-on-3 MoveDownfield with standard RL and skill transfer (ST) from 3-on-2 MoveDownfield (MD).

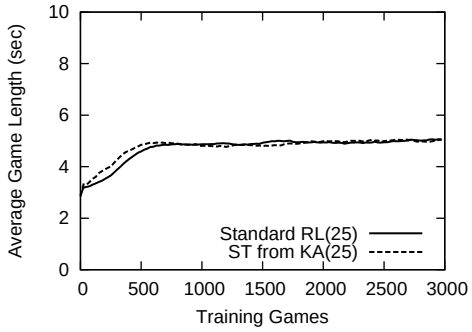

Fig. 12. Average game length while training in 4-on-3 KeepAway with standard RL and skill transfer (ST) from 3-on-2 KeepAway (KA).

Table 5. Statistical results from skill transfer (ST) experiments in BreakAway (BA), MoveDownfield (MD), and KeepAway (KA), comparing area under the curve to standard reinforcement learning (SRL).

Experiment	*Conclusion*	*p-value*	*95% confidence interval*
BA to BA	ST higher with 99% confidence	0.0003	63.75, 203.36
MD to BA	ST higher with 99% confidence	< 0.0001	153.63, 278.02
KA to BA	ST higher with 97% confidence	< 0.0001	176.42, 299.87
MD to MD	ST higher with 98% confidence	< 0.0001	3682.59, 6436.61
KA to KA	ST and SRL equivalent	0.1491	-114.32, 389.20

7 Further Analysis of Skill Transfer

In this section we discuss a few additional experiments that test the boundaries of skill transfer. In particular, we consider the impact of two factors on the effectiveness of skill transfer: quality of learning in the source task, and quality of user guidance.

So far we have performed transfer without paying any attention to the source-task learning curve. However, it might be interesting to know if some types of source-task learning curves indicate better or worse transfer. If we had a choice of source runs to choose from, we could choose one that we expect to produce better target-task performance.

Figure 13 plots the average area under the curve in the target task with skill transfer against the area under the curve in the source task from which transfer was performed. In order to plot data from all the skill-transfer experiments on one scale, we normalize the areas within each group to fall into [0,1]. The correlation coefficient is 0.21, which indicates a small correlation between source-task and target-task area. Therefore it may be helpful to choose source runs with higher area under the curve, although the impact is not likely to be large.

The second factor we consider is how the quality of user guidance affects skill transfer. So far we have simply given reasonable, non-optimized user advice for skill-transfer experiments. Now we investigate the results produced using reasonable variants that a user might easily have used instead.

Figure 14 shows learning curves for skill transfer from KeepAway to BreakAway with variants on the user advice. Variant 1 is more ambitious, encouraging the players to shoot more often, and Variant 2 is more cautious, giving stricter conditions for shooting. These variants are listed in the Appendix (Section A.3).

Figure 15 shows a learning curve for skill transfer from MoveDownfield to BreakAway with no user advice at all. Skill transfer still produces a significantly higher area under the curve than standard RL does, although the gain is smaller. The addition of user advice produces another significant gain.

These results indicate that changes in the user advice do affect the performance of skill transfer, but that reasonable variants still do well. This robustness

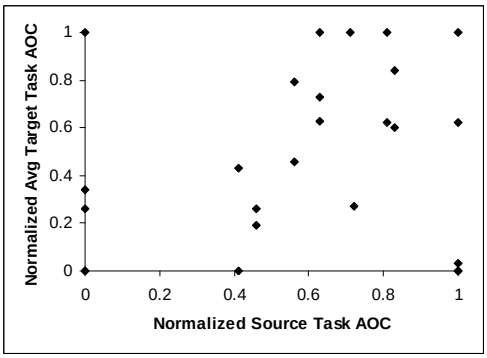

Fig. 13. A plot showing how target-task performance correlates with source-task performance.

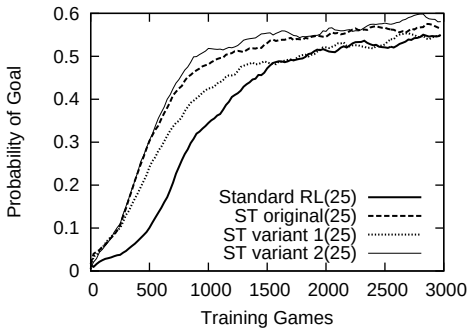

Fig. 14. Probability of scoring a goal while training in 3-on-2 BreakAway with skill transfer from 3-on-2 KeepAway, using variants on the original user advice.

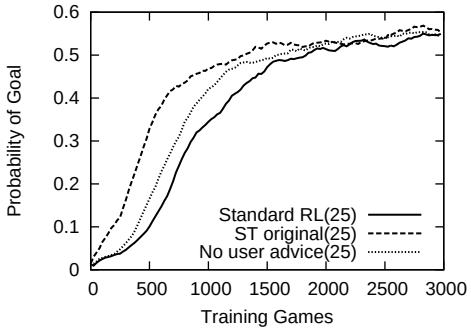

Fig. 15. Probability of scoring a goal while training in 3-on-2 BreakAway with skill transfer from 3-on-2 MoveDownfield, with and without the original user advice.

means that users need not worry about providing perfect advice in order for the skill-transfer method to work. Furthermore, even approximate user advice can significantly improve the performance.

8 Future Work

It may be possible to combine multiple transfer methods to achieve higher performance. Policy transfer and skill transfer could be combined in any scenario, and model reuse could be combined with either for close-transfer scenarios.

We plan to extend our ILP methods to learning multiple-step relational plans instead of single-step rules. This would produce temporally extended actions, which are sometimes known as *options* or *macros*. Whether applied to a target task as advice or through some other approach, relational plans might capture more information from the source task than single rules do.

Another possible direction for this work would be developing advice-taking methods for relational reinforcement learning (RRL). In RRL, as developed by (for example) Dzeroski et al. [8] and Asgharbeygi et al. [3], transfer advice could be applied directly in a first-order form.

9 Conclusions

Reinforcement learners can benefit significantly from knowledge transferred from a previous task. Advice-taking transfer methods, and particularly the ILP-based skill-transfer method, can lead to robust transfer in challenging, dissimilar tasks. The use of first-order logic and relational information helps to separate general from specific information in the source task. The use of advice provides protection against negative transfer effects. Our skill transfer experiments also demonstrate that simple user guidance can naturally be incorporated into advice-based transfer methods, resulting in better transfer.

10 Acknowledgements

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

We offer a public distribution of our RoboCup players and server code at http://www.biostat.wisc.edu/ ml-group/RoboCup.

References

1. D. Andre and S. Russell. Programmable reinforcement learning agents. In *NIPS 13*, 2001.
2. M. Asadi, V. Papudesi, and M. Huber. Learning skill and representation hierarchies for effective control knowledge transfer. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

3. N. Asgharbeygi, D. Stracuzzi, and P. Langley. Relational temporal difference learning. In *ICML*, 2006.
4. J. Clouse and P. Utgoff. A teaching method for reinforcement learning. In *ICML*, 1992.
5. P. Cohen. Personal communication, 2006.
6. K. Driessens and S. Dzeroski. Integrating experimentation and guidance in relational reinforcement learning. In *ICML*, 2002.
7. K. Driessens, J. Ramon, and T. Croonenborghs. Transfer learning for reinforcement learning through goal and policy parametrization. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
8. S. Dzeroski, L. De Raedt, and H. Blockeel. Relational reinforcement learning. In *ICML*, 1998.
9. B. Falkenhainer, K. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63, 1989.
10. K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
11. F. Fernandez and M. Veloso. Policy reuse for transfer learning across tasks with different state and action spaces. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
12. F. Fernandez and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *AAMAS*, 2006.
13. D. Gordon and D. Subramanian. A multistrategy learning scheme for agent knowledge acquisition. *Informatica*, 17:331–346, 1994.
14. N. Gorski and J. Laird. Experiments in transfer across multiple learning mechanisms. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
15. G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *ICML*, 2006.
16. 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.
17. L. Lin. Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8:293–321, 1992.
18. Y. Liu and P. Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *AAAI*, 2006.
19. R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.
20. R. Maclin, J. Shavlik, L. Torrey, and T. Walker. Knowledge-based support vector regression for reinforcement learning. In *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.
21. 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 *AAAI*, 2005.
22. R. Maclin, J. Shavlik, T. Walker, and L. Torrey. A simple and effective method for incorporating advice into kernel methods. In *AAAI*, 2006.
23. M. Madden and T. Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *AI Review 21*, pages 375–398, 2004.
24. O. Mangasarian, J. Shavlik, and E. Wild. Knowledge-based kernel approximation. *JMLR 5*, pages 1127–1141, 2004.

25. N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern. Transfer in variable-reward hierarchical reinforcement learning. In *NIPS Workshop on Transfer Learning*, 2005.
26. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming 19,20*, pages 629–679, 1994.
27. I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
28. T. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, 1999.
29. B. Price and C. Boutilier. Implicit imitation in multiagent reinforcement learning. In *ICML*, 1999.
30. C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *ICML*, 1992.
31. A. Sherstov and P. Stone. Action-space knowledge transfer in MDP's: Formalism, suboptimality bounds, and algorithms. In *COLT*, 2005.
32. S. Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4):323–339, 1992.
33. V. Soni and S. Singh. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI*, 2006.
34. A. Srinivasan. The Aleph manual, 2001.
35. P. Stone and R. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *ICML*, 2001.
36. D. Stracuzzi and N. Asgharbeygi. Transfer of knowledge structures with relational temporal difference learning. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
37. R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning 3*, pages 9–44, 1988.
38. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
39. M. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *AAMAS*, 2005.
40. M. Taylor, P. Stone, and Y. Liu. Value functions for rl-based behavior transfer: A comparative study. In *AAAI*, 2005.
41. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational skill transfer via advice taking. In *ECML*, 2006.
42. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational skill transfer via advice taking. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
43. L. Torrey, T. Walker, J. Shavlik, and R. Maclin. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *ECML*, 2005.
44. T. Walsh, L. Li, and M. Littman. Transferring state abstractions between mdps. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.
45. C. Watkins. Learning from delayed rewards. Technical Report PhD Thesis, University of Cambridge, Psychology Dept., 1989.

Appendix

This appendix gives specifics on mappings, user advice, and skill concepts learned.

A.1 Mappings for Model Reuse and Policy Transfer

Mappings for model reuse and policy transfer match propositional features and actions between the source and target.

For 2-on-1 BreakAway to 3-on-2 BreakAway, we use multiple mappings so that knowledge about the *pass(a1)* action transfers to both *pass(a1)* and *pass(a2)*. Since this is close transfer, the feature and action sets are very similar and straightforward to map. These mappings are shown in Table 6.

Table 6. The two feature and action mappings from 2-on-1 BreakAway to 3-on-2 BreakAway used in model reuse and policy transfer. The less obvious parts are shown in **bold** text.

2-on-1 BreakAway	*3-on-2 BreakAway*
distBetween(a0, a1) distBetween(a0, goalie) **distBetween(a1, goalie)** **angleDefinedBy(a1, a0, goalie)** distBetween(a0, goalLeft) distBetween(a0, goalRight) distBetween(a0, goalCenter) angleDefinedBy(goalLeft, a0, goalie) angleDefinedBy(goalRight, a0, goalie) angleDefinedBy(goalCenter, a0, goalie) angleDefinedBy(topRightCorner, goalCenter, a0) timeLeft moveAhead moveAway moveRight moveLeft shoot(goalLeft) shoot(goalRight) shoot(goalCenter) **pass(a1)**	**distBetween(a0, a1)** distBetween(a0, goalie) **distBetween(a1, goalie)** **angleDefinedBy(a1, a0, goalie)** distBetween(a0, goalLeft) distBetween(a0, goalRight) distBetween(a0, goalCenter) angleDefinedBy(goalLeft, a0, goalie) angleDefinedBy(goalRight, a0, goalie) angleDefinedBy(goalCenter, a0, goalie) angleDefinedBy(topRightCorner, goalCenter, a0) timeLeft moveAhead moveAway moveRight moveLeft shoot(goalLeft) shoot(goalRight) shoot(goalCenter) **pass(a1)**
distBetween(a0, a1) distBetween(a0, goalie) **distBetween(a1, goalie)** **angleDefinedBy(a1, a0, goalie)** distBetween(a0, goalLeft) distBetween(a0, goalRight) distBetween(a0, goalCenter) angleDefinedBy(goalLeft, a0, goalie) angleDefinedBy(goalRight, a0, goalie) angleDefinedBy(goalCenter, a0, goalie) angleDefinedBy(topRightCorner, goalCenter, a0) timeLeft **pass(a1)**	**distBetween(a0, a2)** distBetween(a0, goalie) **distBetween(a2, goalie)** **angleDefinedBy(a2, a0, goalie)** distBetween(a0, goalLeft) distBetween(a0, goalRight) distBetween(a0, goalCenter) angleDefinedBy(goalLeft, a0, goalie) angleDefinedBy(goalRight, a0, goalie) angleDefinedBy(goalCenter, a0, goalie) angleDefinedBy(topRightCorner, goalCenter, a0) timeLeft **pass(a2)**

Table 7. The two feature and action mappings from 3-on-2 MoveDownfield to 4-on-3 MoveDownfield used in model reuse and policy transfer. The less obvious parts are shown in **bold** text.

3-on-2 MoveDownfield	*4-on-3 MoveDownfield*
distBetween(a0, a1)	distBetween(a0, a1)
distBetween(a0, a2)	**distBetween(a0, a3)**
distBetween(a0, d0)	distBetween(a0, d0)
distBetween(a0, d1)	distBetween(a0, d2)
distBetween(a1, minDistDefender(a1))	distBetween(a1, minDistDefender(a1))
distBetween(a2, minDistDefender(a2))	**distBetween(a3, minDistDefender(a3))**
angleDefinedBy(a1, a0, minAngleDefender(a1))	angleDefinedBy(a1, a0, minAngleDefender(a1))
angleDefinedBy(a2, a0, minAngleDefender(a2))	**angleDefinedBy(a3, a0, minAngleDefender(a3))**
distToRightEdge(a0)	distToRightEdge(a0)
distToRightEdge(a1)	distToRightEdge(a1)
distToRightEdge(a2)	**distToRightEdge(a3)**
timeLeft	timeLeft
moveAhead	moveAhead
moveAway	moveAway
moveRight	moveRight
moveLeft	moveLeft
pass(a1)	pass(a1)
pass(a2)	**pass(a3)**
distBetween(a0, a1)	distBetween(a0, a1)
distBetween(a0, a2)	**distBetween(a0, a2)**
distBetween(a0, d0)	distBetween(a0, d0)
distBetween(a0, d1)	distBetween(a0, d2)
distBetween(a1, minDistDefender(a1))	distBetween(a1, minDistDefender(a1))
distBetween(a2, minDistDefender(a2))	**distBetween(a2, minDistDefender(a2))**
angleDefinedBy(a1, a0, minAngleDefender(a1))	angleDefinedBy(a1, a0, minAngleDefender(a1))
angleDefinedBy(a2, a0, minAngleDefender(a2))	**angleDefinedBy(a2, a0, minAngleDefender(a2))**
distToRightEdge(a0)	distToRightEdge(a0)
distToRightEdge(a1)	distToRightEdge(a1)
distToRightEdge(a2)	**distToRightEdge(a2)**
timeLeft	timeLeft
pass(a2)	**pass(a2)**

The mappings from 3-on-2 MoveDownfield to 4-on-3 MoveDownfield and from 3-on-2 KeepAway to 4-on-3 KeepAway are similar in nature. By default, we map the furthest teammate in the source to the furthest teammate in the target, but in a secondary mapping we also cover the second furthest teammate. These mappings are shown in Tables 7 and 8.

For 3-on-2 MoveDownfield to 3-on-2 BreakAway, we use a single mapping that relates players one-to-one. Since this is distant transfer, more features and actions are left out of the mapping. We also chose to leave some move actions out because we felt they might not transfer well in this scenario. This mapping is shown in Table 9.

The mapping for 3-on-2 KeepAway to 3-on-2 BreakAway has even fewer features and actions in common, so that some features in KeepAway are mapped to constants in BreakAway. We chose constants that we thought were typical values for those features. This mapping is shown in Table 10.

Table 8. The two feature and action mappings from 3-on-2 KeepAway to 4-on-3 KeepAway used in model reuse and policy transfer. The less obvious parts are shown in **bold** text.

3-on-2 KeepAway	*4-on-3 KeepAway*
distBetween(k0, k1)	distBetween(k0, k1)
distBetween(k0, k2)	**distBetween(k0, k3)**
distBetween(k0, t0)	distBetween(k0, t0)
distBetween(t0, t1)	distBetween(k0, t2)
distBetween(k1, minDistKeeper(k1))	distBetween(k1, minDistKeeper(k1))
distBetween(k2, minDistKeeper(k2))	**distBetween(k3, minDistKeeper(k3))**
angleDefinedBy(k1, k0, minAngleKeeper(k1))	angleDefinedBy(k1, k0, minAngleKeeper(k1))
angleDefinedBy(k2, k0, minAngleKeeper(k2))	**angleDefinedBy(k3, k0, minAngleKeeper(k3))**
distBetween(k0, fieldCenter)	distBetween(k0, fieldCenter)
distBetween(k1, fieldCenter)	distBetween(k1, fieldCenter)
distBetween(k2, fieldCenter)	**distBetween(k3, fieldCenter)**
distBetween(t0, fieldCenter)	distBetween(t0, fieldCenter)
distBetween(t1, fieldCenter)	distBetween(t2, fieldCenter)
holdBall	holdBall
pass(k1)	pass(k1)
pass(k2)	**pass(k3)**
distBetween(k0, k1)	distBetween(k0, k1)
distBetween(k0, k2)	**distBetween(k0, k2)**
distBetween(k0, t0)	distBetween(k0, t0)
distBetween(t0, t1)	distBetween(k0, t2)
distBetween(k1, minDistKeeper(k1))	distBetween(k1, minDistKeeper(k1))
distBetween(k2, minDistKeeper(k2))	**distBetween(k2, minDistKeeper(k2))**
angleDefinedBy(k1, k0, minAngleKeeper(k1))	angleDefinedBy(k1, k0, minAngleKeeper(k1))
angleDefinedBy(k2, k0, minAngleKeeper(k2))	**angleDefinedBy(k2, k0, minAngleKeeper(k2))**
distBetween(k0, fieldCenter)	distBetween(k0, fieldCenter)
distBetween(k1, fieldCenter)	distBetween(k1, fieldCenter)
distBetween(k2, fieldCenter)	**distBetween(k2, fieldCenter)**
distBetween(t0, fieldCenter)	distBetween(t0, fieldCenter)
distBetween(t1, fieldCenter)	distBetween(t2, fieldCenter)
pass(k2)	**pass(k2)**

Table 9. The feature and action mapping from 3-on-2 MoveDownfield to 3-on-2 BreakAway used in model reuse and policy transfer. The less obvious parts are shown in **bold** text.

3-on-2 MoveDownfield	*3-on-2 BreakAway*
distBetween(a0, a1)	distBetween(a0, a1)
distBetween(a0, a2)	distBetween(a0, a2)
distBetween(a0, d0)	distBetween(a0, d0)
distBetween(a0, d1)	distBetween(a0, d0)
distBetween(a1, minDistDefender(a1))	distBetween(a1, minDistDefender(a1))
distBetween(a2, minDistDefender(a2))	distBetween(a2, minDistDefender(a2))
angleDefinedBy(a1, a0, minAngleDefender(a1))	angleDefinedBy(a1, a0, minAngleDefender(a1))
angleDefinedBy(a2, a0, minAngleDefender(a2))	angleDefinedBy(a2, a0, minAngleDefender(a2))
distToRightEdge(a0)	**distBetween(a0, goalCenter)**
distToRightEdge(a1)	**distBetween(a1, goalCenter)**
distToRightEdge(a2)	**distBetween(a2, goalCenter)**
timeLeft	timeLeft
moveAhead	moveAhead
pass(a1)	pass(a1)
pass(a2)	pass(a2)

Table 10. The feature and action mapping from 3-on-2 KeepAway to 3-on-2 BreakAway used in model reuse and policy transfer. The less obvious parts are shown in **bold** text.

3-on-2 KeepAway	*3-on-2 BreakAway*
distBetween(k0, k1)	distBetween(a0, a1)
distBetween(k0, k2)	distBetween(a0, a2)
distBetween(k0, t0)	distBetween(a0, d0)
distBetween(t0, t1)	distBetween(a0, d0)
distBetween(k1, minDistTaker(k1))	distBetween(a1, minDistDefender(a1))
distBetween(k2, minDistTaker(k2))	distBetween(a2, minDistDefender(a2))
angleDefinedBy(k1, k0, minAngleTaker(k1))	angleDefinedBy(a1, a0, minAngleDefender(a1))
angleDefinedBy(k2, k0, minAngleTaker(k2))	angleDefinedBy(a2, a0, minAngleDefender(a2))
distBetween(k0, fieldCenter)	**15**
distBetween(k1, fieldCenter)	**15**
distBetween(k2, fieldCenter)	**15**
distBetween(t0, fieldCenter)	**10**
distBetween(t1, fieldCenter)	**10**
pass(k1)	pass(a1)
pass(k2)	pass(a2)

A.2 Mappings for Skill Transfer

Mappings for skill transfer match objects in the source and target tasks. In RoboCup, the objects are players and goal parts. We only use single mappings here, because most objects in rules are variables and cover all players.

For 2-on-1 BreakAway to 3-on-2 BreakAway, the mapping is very straightforward:

a0	$\longrightarrow$	a0
a1	$\longrightarrow$	a1
goalie	$\longrightarrow$	goalie
goalLeft	$\longrightarrow$	goalLeft
goalRight	$\longrightarrow$	goalRight
goalCenter	$\longrightarrow$	goalCenter

For 3-on-2 MoveDownfield to 4-on-3 MoveDownfield, it is similar except that the indices of the furthest players change:

a0	$\longrightarrow$	a0
a1	$\longrightarrow$	a1
a2	$\longrightarrow$	a3
d0	$\longrightarrow$	d0
d1	$\longrightarrow$	d2
minDistDefender(a1)	$\longrightarrow$	minDistDefender(a1)
minDistDefender(a2)	$\longrightarrow$	minDistDefender(a3)
minAngleDefender(a1)	$\longrightarrow$	minAngleDefender(a1)
minAngleDefender(a2)	$\longrightarrow$	minAngleDefender(a3)

For 3-on-2 KeepAway to 4-on-3 KeepAway the indices also change:

k0	$\longrightarrow$	k0
k1	$\longrightarrow$	k1
k2	$\longrightarrow$	k3
t0	$\longrightarrow$	t0
t1	$\longrightarrow$	t2
minDistTaker(k1)	$\longrightarrow$	minDistTaker(k1)
minDistTaker(k2)	$\longrightarrow$	minDistTaker(k3)
minAngleTaker(k1)	$\longrightarrow$	minAngleTaker(k1)
minAngleTaker(k2)	$\longrightarrow$	minAngleTaker(k3)

For 3-on-2 MoveDownfield to 3-on-2 BreakAway, there is one less defender:

a0	$\longrightarrow$	a0
a1	$\longrightarrow$	a1
a2	$\longrightarrow$	a2
d0	$\longrightarrow$	d0
d1	$\longrightarrow$	d0
minDistDefender(a1)	$\longrightarrow$	minDistDefender(a1)
minDistDefender(a2)	$\longrightarrow$	minDistDefender(a2)
minAngleDefender(a1)	$\longrightarrow$	minAngleDefender(a1)
minAngleDefender(a2)	$\longrightarrow$	minAngleDefender(a2)

For 3-on-2 KeepAway to 3-on-2 BreakAway, again there is one less defender:

k0	$\longrightarrow$	a0
k1	$\longrightarrow$	a1
k2	$\longrightarrow$	a2
t0	$\longrightarrow$	d0
t1	$\longrightarrow$	d0
minDistTaker(k1)	$\longrightarrow$	minDistDefender(a1)
minDistTaker(k2)	$\longrightarrow$	minDistDefender(a2)
minAngleTaker(k1)	$\longrightarrow$	minAngleDefender(a1)
minAngleTaker(k2)	$\longrightarrow$	minAngleDefender(a2)

A.3 User advice for Skill Transfer

We use a subset of the following user advice in all of our skill transfer experiments:

IF	distBetween(a0, GoalRight) < 10
AND	angleDefinedBy(GoalRight, a0, goalie) > 40
THEN	prefer shoot(GoalRight) over all actions
IF	distBetween(a0, GoalLeft) < 10
AND	angleDefinedBy(GoalLeft, a0, goalie) > 40
THEN	prefer shoot(GoalLeft) over all actions
IF	distBetween(a0, goalCenter) > 10
THEN	prefer moveAhead over moveAway and shoot
Add to pass(Teammate):	diffGoalDistance(Teammate, Value), Value ≥ 1

For each scenario, we only add the user advice that covers skills not already being transferred. We use all of it when transferring from KeepAway to BreakAway, but only the *shoot* parts from MoveDownfield to BreakAway.

The variants of the KeepAway to BreakAway advice that we use in Section 7 are shown below. Variant 1 encourages more attempted shots, which a user might choose under the assumption that more attempts will lead to more success:

IF	distBetween(a0, GoalRight) < 15
AND	angleDefinedBy(GoalRight, a0, goalie) > 35
THEN	prefer shoot(GoalRight) over all actions
IF	distBetween(a0, GoalLeft) < 15
AND	angleDefinedBy(GoalLeft, a0, goalie) > 35
THEN	prefer shoot(GoalLeft) over all actions
IF	distBetween(a0, goalCenter) > 15
THEN	prefer moveAhead over moveAway and shoot
Add to pass(Teammate):	diffGoalDistance(Teammate, Value), Value ≥ 1

Variant 2 does the opposite and gives stricter conditions for shooting, which a user might choose under the assumption that fewer but safer attempts will lead to more success:

IF	distBetween(a0, GoalRight) < 5
AND	angleDefinedBy(GoalRight, a0, goalie) > 45
THEN	prefer shoot(GoalRight) over all actions
IF	distBetween(a0, GoalLeft) < 5
AND	angleDefinedBy(GoalLeft, a0, goalie) > 45
THEN	prefer shoot(GoalLeft) over all actions
IF	distBetween(a0, goalCenter) > 5
THEN	prefer moveAhead over moveAway and shoot
Add to pass(Teammate):	diffGoalDistance(Teammate, Value), Value ≥ 1

A.4 Sample Skill Concepts Learned in Skill Transfer

Below are a few examples of skills that our algorithm learned for skill transfer.

From 2-on-1 BreakAway, an example rule for *shoot* is:

shoot(GoalPart) :-
distBetween(a0, goalCenter) ≥ 6,
angleDefinedBy(GoalPart, a0, goalie) ≥ 52,
distBetween(a0, oppositePart(GoalPart)) ≥ 6,
angleDefinedBy(oppositePart(GoalPart), a0, goalie) ≤ 33,
angleDefinedBy(goalCenter, a0, goalie) ≥ 28.

This rule requires a large open shot angle, a minimum distance to the goal, and angle constraints that restrict the goalie's position to a small area.

From 3-on-2 MoveDownfield, an example rule for *pass* is:

pass(Teammate) :-
distBetween(a0, Teammate) $\geq$ 15,
distBetween(a0, Teammate) $\leq$ 27,
angleDefinedBy(Teammate, a0, minAngleDefender(Teammate)) $\geq$ 24,
distToRightEdge(Teammate) $\leq$ 10,
distBetween(a0, Opponent) $\geq$ 4.

This rule specifies an acceptable range for the distance to the receiving teammate and a minimum pass angle. It also requires that the teammate be close to the finish line on the field and that an opponent not be close enough to intercept.

From 3-on-2 KeepAway, an example rule for *pass* is:

pass(Teammate) :-
distBetween(Teammate, fieldCenter) $\geq$ 6,
distBetween(Teammate, minDistTaker(Teammate)) $\geq$ 8,
angleDefinedBy(Teammate, a0, minAngleTaker(Teammate)) $\geq$ 41,
angleDefinedBy(OtherTeammate, a0, minAngleTaker(OtherTeammate)) $\leq$ 23.

This rule specifies a minimum pass angle and an open distance around the receiving teammate. It also requires that the teammate not be too close to the center of the field and gives a maximum pass angle for the alternate teammate.

Some parts of these rules were unexpected, but make sense in hindsight. For example, the shoot rule specifies a minimum distance to the goal rather than a maximum distance. Presumably this is because large shot angles are only available at reasonable distances anyway. This shows the advantages that advice learned through transfer can have over user advice.