Transfer in Reinforcement Learning via Markov Logic Networks

Lisa Torrey, Jude Shavlik, Sriraam Natarajan, Pavan Kuppili, Trevor Walker
Computer Sciences Department
University of Wisconsin-Madison

Abstract

We propose the use of statistical relational learning, and in particular the formalism of Markov Logic Networks, for transfer in reinforcement learning. Our goal is to extract relational knowledge from a source task and use it to speed up learning in a related target task. We do so by learning a Markov Logic Network that describes the source-task *Q*-function, and then using it for decision making in the early learning stages of the target task. Through experiments in the RoboCup simulated-soccer domain, we show that this approach can provide a substantial performance benefit in the target task.

Introduction

Human learners appear to have inherent ways to transfer knowledge between tasks. That is, we recognize and apply relevant knowledge from previous learning experiences when we encounter new tasks. The more related a new task is to our previous experience, the more quickly we can master it.

Common machine learning algorithms, in contrast, usually address isolated tasks. Transfer learning research attempts to develop methods to transfer knowledge learned in a source task and use it to speed up learning in a related target task. Algorithms that enable knowledge transfer represent progress towards making machine learning as effective as human learning.

One area in which transfer is often desirable is reinforcement learning (RL), since standard RL algorithms can require long training times in complex domains. One example of a complex RL domain is the simulated soccer project RoboCup (Noda et al. 1998), which we use for experiments in this work. Our work focuses on *relational* transfer in domains like RoboCup, where the source-task knowledge includes first-order logical information. Here we propose the use of statistical relational learning for transfer in RL.

Statistical relational learning (SRL) is a type of machine learning designed to operate in domains that have both uncertainty and rich relational structure. It focuses on combining the two powerful paradigms of first-order logic, which generalizes among the objects in a domain, and probability theory, which handles uncertainty. One recent and popular SRL formalism is the Markov Logic Network (MLN), introduced by Richardson and Domingos (2005), which interprets first-order statements as soft constraints with weights. This framework can allow us to express source-task knowledge with relational structure while still accounting for nondeterministic events.

Our approach is to learn an MLN that describes the source-task *Q*-function and then use this function for decision making in the early learning stages of the target task. Doing so provides an initial bias towards behavior that is likely to be successful, and avoids the slow process of random exploration that traditionally occurs at the beginning of RL. This method takes advantage of the relational structure of a domain by generalizing across objects even if the RL algorithm itself does not, as is the case in domains like RoboCup where fully relational RL is not yet achieved. It can therefore allow generalization to new tasks more easily, since it captures knowledge about logical variables rather than constants.

With experiments in RoboCup, we show that transfer in RL via Markov Logic Networks can provide a substantial head start in a target task.

Reinforcement Learning in RoboCup

In reinforcement learning (Sutton and Barto 1998), an agent navigates through an environment trying to earn rewards or avoid penalties. The environment's state is described by a set of features, and the agent takes actions to cause the state to change.

In one common form called *Q*-learning (Watkins 1989), 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 (possibly zero), the agent updates its *Q*-value estimates for the current state.

Stone and Sutton (2001) introduced RoboCup as an RL domain that is challenging because of its large, continuous state space and non-deterministic action effects. The RoboCup project 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. Since the

Appears in the 2008 AAAI Workshop on Transfer Learning for Complex Tasks.

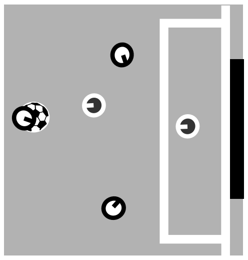

Figure 1: Snapshot of a 3-on-2 BreakAway game. The attacking players have possession of the ball and are maneuvering against the defending team towards the goal.

full game of soccer is quite complex, researchers have developed several simpler games within the RoboCup simulator. See Figure 1 for a snapshot of one of these games.

In M-on-N BreakAway (Torrey et al. 2005), the objective of the M reinforcement learners called *attackers* is to score a goal against $N-1$ hand-coded *defenders* and a *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 0 reward otherwise. The attacker who has the ball may choose to move (ahead, away, left, or right with respect to the goal center), pass to a teammate, or shoot (at the left, right, or center part of the goal).

RoboCup tasks are inherently multi-agent games, but a standard simplification is to have only one learning agent. This agent controls the attacker currently in possession of the ball, switching its "consciousness" between attackers as the ball is passed. Attackers without the ball follow simple hand-coded policies that position them to receive passes.

Table 1 shows the state features for BreakAway, which mainly consist of distances and angles between players and the goal. They are represented in logical notation, though our RL algorithm uses the grounded versions of these predicates in a fixed-length feature vector. Capitalized atoms indicate typed variables, while constants and predicates are uncapitalized. The attackers (labeled *a0*, *a1*, etc.) are ordered by their distance to the agent in possession of the ball (*a0*), as are the non-goalie defenders (*d0*, *d1*, etc.).

Our RL implementation uses a $SARSA(\lambda)$ variant of *Q*-learning (Sutton 1988) and employs a support vector machine for function approximation (Maclin et al. 2005). We relearn the *Q*-function after every batch of 25 games. The exploration rate begins at 2.5% and decays exponentially over time. Stone and Sutton (2001) found that discretizing the continuous features into Boolean interval features called *tiles* is useful for learning in RoboCup; following this approach we create 32 tiles per feature.

Agents in the games of 2-on-1 and 3-on-2 BreakAway take between 1000 and 3000 training episodes to reach a performance asymptote in our system. These games are similar, but their differences in the numbers of attackers and defenders cause substantial differences in their optimal policies, particularly since there is an entirely new type of player in 3-on-2 (the non-goalie defender). Despite the differences, the tasks do have the same objective and can be expected to require some similar strategic aspects.

Table 1: The features that describe a BreakAway state.

distBetween(a0, Player)
distBetween(a0, GoalPart)
distBetween(Attacker, goalCenter)
distBetween(Attacker, ClosestDefender)
distBetween(Attacker, goalie)
angleDefinedBy(topRight, goalCenter, a0)
angleDefinedBy(GoalPart, a0, goalie)
angleDefinedBy(Attacker, a0, ClosestDefender)
angleDefinedBy(Attacker, a0, goalie)
timeLeft

Related Work in Transfer Learning

The goal in transfer learning is to speed up learning in a target task by transferring knowledge from a related source task. One straightforward way to do this in reinforcement learning is to begin performing the target task using the source-task value functions, after a suitable mapping between features and actions in the source and target tasks. Taylor, Stone and Liu (2005) apply this type of transfer method in the RoboCup domain. Our MLN transfer method is related to value-function transfer, but it incorporates relational information into the transferred function, transferring knowledge about logical variables rather than constants.

Another approach that has been proposed 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 referred to as *policy reuse* (Fernandez and Veloso 2006)). Our MLN transfer method also replaces random exploration with source-task policy decisions, but it does so more aggressively, and it also enhances the source-task policy with relational knowledge.

Our previous work includes *skill transfer* (Torrey et al. 2006), in which we use inductive logic programming to learn rules that indicate when the agent chooses to take source-task actions. We use those rules as *advice* in the target task, placing soft constraints on the solution that can be followed or ignored according to how successful they are. Taylor and Stone (2007) also learn a set of rules for taking actions, and they use different advice-taking mechanisms: for example, they give a *Q*-value bonus to the advised action. Our MLN transfer method moves away from the advice-taking paradigm to a more direct use of transferred knowledge.

Another approach in our previous work involves transferring *relational macros* (Torrey et al. 2007). A relational macro is a finite-state machine representing an action plan in which decisions are made in first-order logic. We use inductive logic programming to learn a macro that characterizes successful behavior in the source task, and then use the macro for decision-making in the early learning stages of the target task.

Our MLN transfer method is related to macro transfer, particularly since it uses the same technique of demonstration in the target task, but it transfers an entire relational *Q*-function instead of a single plan. It also offers better-defined opportunities for refinement of transferred knowledge in the target task, since MLNs are more well-known and researched models than the macro structures we designed. Refinement in the target task is proposed in the context of incremental relational regression trees by Ramon, Driessens, and Croonenborghs (2007).

Relational reinforcement learning (RRL) itself can be considered a related topic (Tadepalli, Givan, and Driessens 2004). In RRL, state descriptions and learned models use first-order logic, which provides opportunities for transferring relational concepts. Croonenborghs, Driessens, and Bruynooghe (2007) learn relational options for use in relational RL. Fully relational learning has not yet been achieved in domains as complex as RoboCup; our methods transfer relational knowledge, but do not yet attempt to do relational *Q*-learning in the target task.

Other work involving MLNs in transfer learning includes Mihalkova and Mooney (2006) and Mihalkova, Huynh and Mooney (2007). Their work addresses transfer between classification tasks solved by Markov Logic Networks, whereas ours creates Markov Logic Networks to transfer between reinforcement learning tasks.

Statistical relational models have also been used to specify prior knowledge in RL. Natarajan, Tadepalli and Fern (2007) create relational hierarchies of prior knowledge to help an agent learn to assist a user. Our MLN transfer method applies similar ideas in the transfer setting.

Transferring a Markov Logic Network

A Markov network can be viewed as a set of ground predicates with potential functions that define a probability distribution over possible worlds. A Markov Logic Network (Richardson and Domingos 2005) is a set of first-order logic formulas that can be grounded to form a Markov network. Each formula describes a property that may be present in the world, and has an associated real-valued weight. Worlds become more probable as they satisfy more high-weighted formulas.

Given a set of formulas along with positive and negative examples of worlds, the weights can be learned via gradient descent (Lowd and Domingos 2007). Then, given a set of *evidence* about a world–a list of predicates that are known to be true or false–standard inference in the ground Markov network can determine the probability that the remaining predicates are true or false (Richardson and Domingos 2005).

We use an MLN to define a probability distribution for the *Q*-value of an action, conditioned on the state features. In this scenario, a world corresponds to a state in the RL environment, and a formula describes some characteristic that helps determine the *Q*-value of an action in that state. For example, assume that there is a discrete set of *Q*-values that a RoboCup action can have (*high*, *medium*, and *low*). In this simplified case, formulas in an MLN representing the *Q*-function for BreakAway could look like the following:

levelOfQvalue(move(ahead), high) $\longleftarrow$
    distBetween(a0, GoalPart) > 10,
    angleDefinedBy(GoalPart, a0, goalie) < 30.

The MLN could contain multiple formulas like this for each action, each with a weight learned via gradient descent from a training set of source-task states in which all the properties and *Q*-values are known. We could then use this MLN to evaluate action *Q*-values in a target-task state: we evaluate which properties are present and absent in the state, give that information as evidence, and infer the probability that each action's *Q*-value is high, medium, or low.

Note that *Q*-values in RoboCup are continuous rather than discrete, so we do not actually learn rules classifying them as high, medium, or low. However, we do discretize the *Q*-values into bins, using a procedure described in the next section to find bins that fit the data.

Learning an MLN from a Source Task

During source-task learning with RL, the agent trains a normal *Q*-function after each chunk using data selected from the episodes played so far. In our support-vector machine implementation, which we use for learning in the source task following the experimental methodology of our previous work (Torrey et al. 2005; 2006; 2007), this data includes up to 1000 examples of states and *Q*-values for each action. We choose a source-task chunk and use its data to learn a MLN for transfer (referred to henceforth as the MLN *Q*-function). The choice of the source-task chunk has an impact; we discuss this effect in a later section.

The first step in our procedure is to separate the *Q*-values for an action into bins. The training example *Q*-values could have any arbitrary distribution, so we use the hierarchical clustering algorithm in Table 2 to find good bins. Initially every training example is its own cluster, and we repeatedly join clusters whose midpoints are closest until there are no midpoints closer than ϵ apart. The final cluster midpoints serve as the midpoints of the bins.

The value of ϵ is domain-dependent. For BreakAway, which has *Q*-values ranging from approximately 0 to 1, we use $\epsilon = 0.1$. This leads to a maximum of about 11 bins, but there are often less because training examples tend to be distributed unevenly across the range. We experimented with ϵ values ranging from 0.05 to 0.2 and found very minimal differences in the results; the approach appears to be robust to the choice of ϵ within a reasonably wide range.

Table 2: Our algorithm for creating bins for the *Q*-values of action a.

```
For each training example $i$ for action $a$
    Create cluster $c_i$ containing only the $Q$-value of example $i$
Let $C$ = sorted list of $c_i$ for all $i$
Let $m$ = min distance between two adjacent $c_x, c_y \in C$
While $m < \epsilon$
    Join clusters $c_x$ and $c_y$ into $c_{xy}$
    $C \longleftarrow C \cup c_{xy} - \{c_x, c_y\}$
    $m \longleftarrow$ min distance between two new adjacent $c'_x, c'_y \in C$
$C$ is now the final set of bin midpoints
Place bin dividers halfway between the midpoints
Return bins
```

The next step in our procedure is to learn rules that classify the training examples into the bins. We learn these rules with an inductive logic programming system called Aleph (Srinivasan 2001). The Aleph algorithm selects an example, builds the most specific clause that entails the example, and searches for a clause that covers other examples while maximizing a provided scoring function.

Scoring functions typically involve *precision* and *recall*. The precision of a rule is the fraction of examples it calls positive that are truly positive, and the recall is the fraction of truly positive examples that it correctly calls positive. The scoring function we use is

$$F = \frac{2 * Precision * Recall}{Precision + Recall}$$

because we consider both precision and recall to be important. We use both a heuristic and randomized search algorithm to find potential rules.

We also use a system called Gleaner (Goadrich, Oliphant, and Shavlik 2006) to record rules encountered during the Aleph search that have high precision over varying levels of recall. From these we select a final ruleset to use as formulas in the MLN with the algorithm in Table 3. To ensure high coverage and minimal duplication, we sort the rules by decreasing precision and greedily select them if they improve the F score of the ruleset. (Due to the large number of rules stored by Gleaner, a non-greedy rule selection approach would take prohibitively long to run.)

The final rulesets for all the actions become the set of formulas in the MLN *Q*-function. We typically end up with a few dozen formulas per bin in our experiments.

The final step in our procedure is to learn weights for these formulas. We use the scaled conjugate-gradient algorithm in the Alchemy MLN implementation (Kok et al. 2005) to learn weights.

Applying an MLN in a Target Task

Given an MLN *Q*-function, we can estimate the *Q*-value of an action in a target-task state with the algorithm in Table 4. We begin by performing inference in the MLN to estimate the probability, for each action and bin, that *levelOfQ-value(action, bin)* is true. Typically, inference in MLNs is approximate because exact inference is intractable for most networks, but in our case exact inference is possible because there are no missing features and the Markov blanket of a query node contains only known evidence nodes.

For each action a, we infer the probability p_b that the *Q*-value falls into each bin b. We then use these probabilities as weights in a weighted sum to calculate the *Q*-value of a:

$$Q_a(s) = \sum_b (p_b * E[Q_a|b])$$

where $E[Q_a|b]$ is the expected *Q*-value given that b is the correct bin. We estimate this by the *Q*-value of the training example in the bin that is most similar to state s. This method performed slightly better than taking the average *Q*-value of all the training examples in the bin, which would be another reasonable estimate for the expected *Q*-value.

The similarity measure between two states is calculated by the algorithm in Table 5. It is the dot product of two vectors that indicate which of the bin clauses the states satisfy. For each formula, a vector has a $+1$ entry if the state satisfies it or a -1 entry if it does not.

To use a transferred MLN in a target task, we use the demonstration approach that we used when transferring macros (Torrey et al. 2007). Our target-task learner begins by using the MLN to evaluate *Q*-values for a set of episodes, instead of exploring randomly as an untrained RL agent would traditionally do. An alternate approach might be to continue learning the target task with an MLN *Q*-function, but this remains a direction for future work, since it requires

Table 3: Our algorithm for selecting the final ruleset for a bin. The F scoring function combines precision and recall.

```
Let $R$ = all rules encountered during Aleph search
Let $S$ = $R$ sorted by decreasing precision on the training set
Let $T$ = $\emptyset$
For each rule $r \in S$
    Let $U$ = $T \cup \{r\}$
    If F($U$) > F($T$)
    Then $T \longleftarrow U$
Return final ruleset $T$
```

Table 4: Our algorithm for calculating the Q-value of action a in target-task state s using the MLN Q-function.

```
Provide state $s$ to the MLN as evidence
For each bin $b \in [1, 2, ..., n]$
    Infer the probability $p_b$ that $Q_a(s)$ falls into bin $b$
    Find the training example $t$ in bin $b$ most similar to $s$
    Let $E[Q_a|b] = Q_a(t)$
Return $Q_a(s)$ = $\sum_b (p_b * E[Q_a|b])$
```

Table 5: Our algorithm for measuring similarity between a target-task state s and a training-example state t in bin b.

```
For each rule $i \in [1, 2, ..., r]$ in the final ruleset for bin $b$
    If rule $i$ is satisfied in state $s$
        Then let $v_i(s) = +1$
        Otherwise let $v_i(s) = -1$
    If rule $i$ is satisfied in state $t$
        Then let $v_i(t) = +1$
        Otherwise let $v_i(t) = -1$
Let $V_s = (v_1(s), v_2(s), ..., v_r(s))$
Let $V_t = (v_1(t), v_s(t), ..., v_r(t))$
Return similarity($s, t$) = $V_s \cdot V_t$
```

overcoming considerable challenges in computational complexity.

The demonstration period lasts for 100 games in our system, and after each batch of 25 games we re-learn the *Q*-function via support-vector regression. After 100 games, we continue learning the target task with standard RL. This generates new *Q*-value examples in the standard way, and we combine these with the old MLN-generated examples as we continue re-learning the *Q*-function after each batch. As the new examples accumulate, we gradually drop the old examples by randomly removing them at the rate that new ones are being added.

The demonstration approach carries a risk for negative transfer if the source and target tasks are not similar enough. However, if the user believes that the tasks are similar enough, the potential benefits could outweigh that risk. The approach also has the desirable property of making no assumptions about the learning algorithm in the source or target task.

Since standard RL chooses actions randomly in the early steps of a new task, a good MLN *Q*-function can provide a significant advantage. The performance level of the demonstrated strategy is unlikely to be as high as the target-task agent can achieve with further training, unless the tasks are similar enough to make transfer a trivial problem, but the hope is that the learner can smoothly improve its performance from the level of the demonstration up to its asymptote. If there is limited time and the target task cannot be trained to its asymptote, then the immediate advantage that this approach can provide may be even more valuable.

Experimental Results

We present results from MLN transfer experiments in the RoboCup domain. To test our approach, we learn MLNs from source tasks in 2-on-1 BreakAway and transfer them to 3-on-2 BreakAway. We learn the source tasks with standard RL for 3000 games, and then we train the target tasks for 3000 games to show both the initial advantage of the MLNs and the behavior as training continues.

While performing this evaluation, we discovered that the results can vary widely depending on the source-task chunk from which we transfer. Our transfer methods in previous work, as well as other approaches we have seen, use the last model learned for the source task. However, we found that MLN transfer performs significantly better when we use a model halfway through the source-task learning curve. We therefore adjusted our method to transfer from a chunk near the halfway point in the curve. We believe the reason earlier models transfer better is that they contain more general knowledge, whereas the fully learned models contain more specific knowledge that is less transferrable.

Figure 2 shows the target-task learning curves from these experiments. It compares our approach against normal *Q*-learning in the target task as well as the two most related methods: relational-macro transfer (Torrey et al. 2007) and a version of value-function transfer (Taylor, Stone, and Liu 2005) that we adapted for our batch RL algorithm. Each curve in the figure is an average of 25 runs and has points smoothed over the previous 500 games to account for the high variance in the RoboCup domain. The transfer curves consist of five target-task runs generated from each of five source-task runs, to account for variance in both stages of learning.

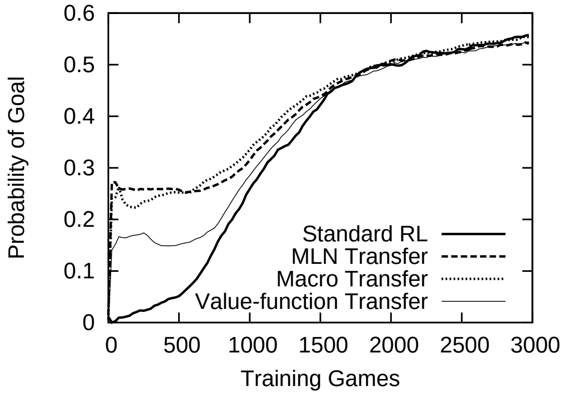

Figure 2: Probability of scoring a goal in 3-on-2 BreakAway, with *Q*-learning and with three transfer approaches that use 2-on-1 BreakAway as the source task.

Our agents in 2-on-1 BreakAway reach a performance asymptote of scoring in approximately 70% of the episodes. The MLNs learned from the 2-on-1 source runs, when used to play new games of 2-on-1 BreakAway, score in approximately 65% of the episodes. In comparison, this self-transfer score for relational macros is only 50%. The MLN transfer method therefore describes the source-task behavior more thoroughly than the macro transfer method does.

However, these additional nuances captured by the MLN do not lead to better transfer in this case: the performance of MLN transfer in these experiments is comparable to that of macro transfer. Both have significantly better early performance than standard *Q*-learning and our implementation of value-function transfer.

Conclusions and Future Work

We propose an algorithm for transfer in reinforcement learning via statistical relational learning. We use a Markov Logic Network to represent the *Q*-function of the source task relationally, and we use this transfer knowledge in the target task to provide a good starting point for training. We motivate the use of MLNs for transfer, outline the methodology for learning and applying an MLN *Q*-function, and evaluate the method on transfer scenarios in the RoboCup domain.

Our results show that MLN transfer can give the learner a significant head start in the target task. The approach performs well in comparison to our implementation of value-function transfer, and it performs comparably to relational-macro transfer.

A potential advantage of MLN transfer over relational-macro transfer, and a possible reason to choose MLNs over macros given their comparable performance, is that it provides better opportunities for theory refinement in the target task. Instead of using a transferred structure for a fixed amount of time and then abandoning it, we could revise the structure as we learn in the target task. Existing work on revision of MLNs, such as Mihalkova, Huynh and Mooney

(2007), could be applied to this problem.

The advantage that MLN transfer has over propositional value-function transfer is that it makes use of the relational information present in the domain. It “lifts” the transferred information to the level of first-order logic, even though reinforcement learning in RoboCup is currently propositional. This makes the transferred knowledge more general and thus more easily applicable in some target tasks. It also makes the mapping between source and target tasks implicit rather than explicit by generalizing over groups of related objects.

Another area for future work is scaling up to fully relational reinforcement learning in RoboCup. Since an MLN represents an entire Q-function, it could serve as a function approximator in RL, or it could approximate a transition function. Some potential advantages of this approach are generalization over objects and intelligent exploration using information about Q-value probability distributions. The main challenge to overcome in performing relational RL with MLNs is the computational cost involved.

One insight these experiments provided is that the source-task chunk from which transfer is performed can have a significant impact in some transfer methods. A fully learned source-task model can transfer less effectively than a partially learned one. We believe this is because the fully learned models can contain more specific knowledge that is less transferrable, and therefore cause a form of overfitting from the perspective of transfer learning. It may be that other approaches should be revisited in light of this observation. In future work, we hope to develop a way to screen models and predict which will transfer better.

Acknowledgements

This research is supported by DARPA grants HR0011-07-C-0060 and FA8650-06-C-7606.

References

Croonenborghs, T.; Driessens, K.; and Bruynooghe, M. 2007. Learning relational skills for inductive transfer in relational reinforcement learning. In *2007 International Conference on Inductive Logic Programming*.

Fernandez, F., and Veloso, M. 2006. Policy reuse for transfer learning across tasks with different state and action spaces. In *2006 ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

Goadrich, M.; Oliphant, L.; and Shavlik, J. 2006. Gleaner: creating ensembles of first-order clauses to improve recall-precision curves. *Machine Learning* 64:231–261.

Kok, S.; Singla, P.; Richardson, M.; and Domingos, P. 2005. The Alchemy system for statistical relational AI. Technical report, University of Washington.

Lowd, D., and Domingos, P. 2007. Efficient weight learning for markov logic networks. In *2007 Conference on Knowledge Discovery in Databases*.

Maclin, R.; Shavlik, J.; Torrey, L.; and Walker, T. 2005. Knowledge-based support vector regression for reinforcement learning. In *2005 IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*.

Mihalkova, L., and Mooney, R. 2006. Transfer learning with Markov logic networks. In *2006 ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

Mihalkova, L.; Huynh, T.; and Mooney, R. 2007. Mapping and revising Markov logic networks for transfer learning. In *2007 AAAI Conference on Artificial Intelligence*.

Natarajan, S.; Tadepalli, P.; and Fern, A. 2007. A relational hierarchical model for decision-theoretic assistance. In *2007 International Conference on Inductive Logic Programming*.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence* 12:233–250.

Ramon, J.; Driessens, K.; and Croonenborghs, T. 2007. Transfer learning in reinforcement learning problems through partial policy recycling. In *2007 European Conference on Machine Learning*.

Richardson, M., and Domingos, P. 2005. Markov logic networks. *Machine Learning* 62:107–136.

Srinivasan, A. 2001. The Aleph manual (available online).

Stone, P., and Sutton, R. 2001. Scaling reinforcement learning toward RoboCup soccer. In *2001 International Conference on Machine Learning*.

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

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

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational reinforcement learning: An overview. In *2004 ICML Workshop on Relational Reinforcement Learning*.

Taylor, M., and Stone, P. 2007. Cross-domain transfer for reinforcement learning. In *2007 International Conference on Machine Learning*.

Taylor, M.; Stone, P.; and Liu, Y. 2005. Value functions for RL-based behavior transfer: A comparative study. In *2005 AAAI Conference on Artificial Intelligence*.

Torrey, L.; Walker, T.; Shavlik, J.; and Maclin, R. 2005. Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *2005 European Conference on Machine Learning*.

Torrey, L.; Shavlik, J.; Walker, T.; and Maclin, R. 2006. Skill acquisition via transfer learning and advice taking. In *2006 European Conference on Machine Learning*.

Torrey, L.; Shavlik, J.; Walker, T.; and Maclin, R. 2007. Relational macros for transfer in reinforcement learning. In *2007 International Conference on Inductive Logic Programming*.

Watkins, C. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, University of Cambridge.