Policy Transfer via Markov Logic Networks

Lisa Torrey and Jude Shavlik

University of Wisconsin, Madison WI, USA
ltorrey@cs.wisc.edu, shavlik@cs.wisc.edu

Abstract. We propose using a statistical-relational model, the Markov Logic Network, for knowledge 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 show that Markov Logic Networks are effective models for capturing both source-task Q-functions and source-task policies. We apply them via demonstration, which involves using them for decision making in an initial stage of the target task before continuing to learn. Through experiments in the RoboCup simulated-soccer domain, we show that transfer via Markov Logic Networks can significantly improve early performance in complex tasks, and that transferring policies is more effective than transferring Q-functions.

1 Introduction

The transfer of knowledge from one task to another is a desirable property in machine learning. Our ability as humans to transfer knowledge allows us to learn new tasks quickly by taking advantage of relationships between tasks. While many machine-learning algorithms learn each new task from scratch, there are also *transfer-learning* algorithms [13] that can improve learning in a *target task* using knowledge from a previously learned *source task*.

In reinforcement learning (RL), an agent navigates through an environment, sensing its state, taking actions, and trying to earn rewards [12]. The *policy* of the agent determines which action it chooses in each step. An agent performing RL typically learns a *value function* to estimate the values of actions as a function of the current state, and its policy typically is to take the highest-valued action in all except occasional *exploration* steps.

In complex domains, RL can require many early episodes of nearly random exploration before acquiring a reasonable value function or policy. A common goal of transfer in RL is to shorten or remove this period of low performance. Recent research has yielded a wide variety of RL transfer algorithms to accomplish this goal [13]. In one category of methods, RL agents apply a source-task policy or value function at some point(s) while learning the target task. Approaches of this type vary in the representation of the source-task knowledge and in the timing and frequency of its application.

Appears in the ILP-2009 Springer LNCS Post-conference Proceedings.

Madden and Howley [6] learn a set of rules to represent a source-task policy, and they use those rules only during exploration steps in the target task. Fernandez and Veloso [2] use the original representation of the source-task policy, and give the target-task agent a three-way choice between using the current target-task policy, using a source-task policy, and exploring randomly. Croonenborghs et al. [1] learn a relational decision tree to represent the source-task policy, and use the tree as a multi-step action (an *option*).

Our own work in this area has contributed several relational methods, in which the knowledge transferred is at the level of first-order logic, and is extracted from the source task with inductive logic programming (ILP). Using ILP [8], we transfer several types of relational models. In one recent approach [15], the transferred model is a first-order finite-state machine that we call a *relational macro*, and it represents a successful generalized source-task plan.

In this paper, we propose transfer via a statistical-relational model called a Markov Logic Network (MLN). An MLN combines first-order logic and probability [9], and is capable of capturing more source-task knowledge than a macro can. With experiments in the RoboCup domain [7], we show that MLN transfer methods can significantly improve initial performance in complex RL tasks.

2 Reinforcement Learning in RoboCup

In one common form of RL called Q-learning [12], the value function learned by the agent is called a Q-function, and it estimates the value of taking an action from a state. The policy is to take the action with the highest Q-value in the current state, except for occasional exploratory actions taken in a small percent ϵ of steps. After taking an action and receiving some reward (possibly zero), the agent updates its Q-value estimates for the current state.

Stone and Sutton [11] introduced RoboCup [7] as an RL domain that is challenging because of its large, continuous state space and non-deterministic action effects. Since the full game of soccer is quite complex, researchers have developed several simpler games within the RoboCup simulator.

In M-on-N BreakAway (see Figure 1), 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).

Figure 2 shows the state features for BreakAway, which mainly consist of distances and angles between players and the goal. They are shown in logical notation since we perform transfer learning in first-order logic; our basic RL algorithm uses grounded literals in a fixed-length feature vector. Capitalized atoms indicate typed variables, while constants and predicates are uncapitalized.

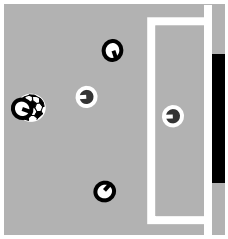

Fig. 1. Snapshot of a 3-on-2 BreakAway game. The attacking players have possession of the ball and are maneuvering against the defending team towards the goal.

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

Fig. 2. The features that describe a BreakAway state in their first-order logical form, where variables are capitalized.

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 basic RL algorithm uses a $SARSA(\lambda)$ variant of Q-learning [12] and employs a support vector machine (SVM) for Q-function approximation [5]. It relearns the SVM Q-function after every batch of 25 games. The exploration rate ϵ begins at 2.5% and decays exponentially over time. Stone and Sutton [11] found that discretizing the continuous features into Boolean interval features called *tiles* is important for learning in RoboCup; following this approach, we add 32 tiles per feature.

Agents in the games of 2-on-1, 3-on-2, and 4-on-3 BreakAway take between 1000 and 3000 training episodes to reach a performance asymptote in our system. Differences in the numbers of attackers and defenders cause substantial differences in optimal policies, particularly since there is a type of player entirely missing in 2-on-1 (the non-goalie defender). However, there remain strong relationships between BreakAway games of different sizes, and transfer between them should improve learning.

3 Markov Logic Networks

The Markov Logic Network (MLN) is a model developed by Richardson and Domingos [9] that combines first-order logic and probability. It expresses concepts with first-order rules, as ILP does, but unlike ILP it puts weights on the rules to indicate how important they are. While ILP rulesets can only predict a concept to be true or false, an MLN can estimate the probability that a concept is true, by comparing the total weight of satisfied rules to the total weight of violated rules. This type of probabilistic logic therefore conveys more information than pure logic. It is also less brittle, since world states that violate some rules are not impossible, just less probable.

Formally, a Markov Logic Network is a set of first-order logic formulas F, with associated real-valued weights W, that provides a template for a Markov

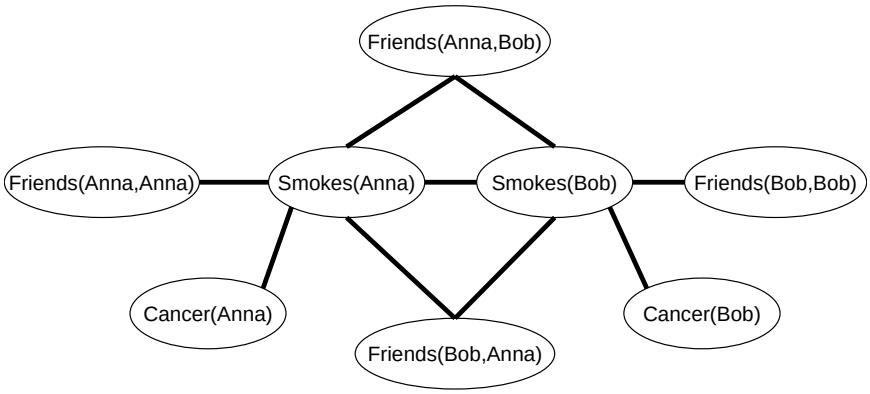

Fig. 3. The ground Markov network produced by the MLN described in this section. This example and this image come from Richardson and Domingos [9]. Each clique in this network has a weight (not shown) derived from the formula weights.

network. The network contains a binary node for each possible grounding of each predicate of each formula in F, with groundings determined by a set of constants C. Edges exist between nodes if they appear together in a possible grounding of a formula. Thus the graph contains a clique for each possible grounding of each formula in F.

The classic example from Richardson and Domingos [9] follows. Suppose the formulas are:

Weight 1.5: $\forall y$ Smokes$(y) \Rightarrow$ Cancer(y)
Weight 0.8: $\forall y, z$ Friends$(y, z) \Rightarrow$ (Smokes$(y) \Leftrightarrow$ Smokes(z))

These rules assert that smoking leads to cancer and that friends have similar smoking habits. These are both good examples of MLN formulas because they are often true, but not always; thus they will have finite weights. Given constants *Anna* and *Bob* that may be substituted for the variables y and z, this MLN produces the ground Markov network in Figure 3. (Note that the convention for capitalization is opposite here from in ILP; variables here are lower-case and constants are upper-case.)

Let X represent all the nodes in this example, and let $X = x$ indicate that among the possible worlds (the true/false settings of those nodes), x is the actual one. The probability distribution represented by the Markov network is:

$$P(X = x) = \frac{1}{Z}\, exp \sum_{i \in F} w_i n_i(x) \tag{1}$$

Here Z is a normalizing term, w_i is the weight of formula $i \in F$, and $n_i(x)$ is the number of true groundings of formula i in x. Based on this equation, one can calculate the probability of any node in the network given *evidence* about the truth values of some other nodes.

Given a set of positive and negative examples of worlds, appropriate formula weights can be learned rather than specified manually. There are several algorithms for weight learning; the current state-of-the-art is a method called *preconditioned scaled conjugate gradient* [4]. This is the default algorithm in the Alchemy software package [3], which we use for our experiments.

We learn formulas with ILP, and then assign weights to them with Alchemy. The type of formulas we learn determines the type of source-task knowledge captured by the MLN. The following sections describe two possible types.

4 MLN Q-Function Transfer

MLN Q-function transfer [14] is a transfer method that learns an MLN to express the source-task Q-function relationally, and allows target-task agents to use it for an initial demonstration period. This allows the target-task agents to avoid the slow process of exploration that traditionally occurs at the beginning of RL.

This method uses an MLN to define a probability distribution for the Q-value of an action, conditioned on the state features. It chooses a source-task batch and uses its training data to learn an MLN Q-function for transfer. The choice of which source-task batch has an impact, as we will discuss.

In this scenario, an MLN formula describes some characteristic of the RL agent's environment 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, one formula in an MLN representing the Q-function for BreakAway could look like the following:

IF distBetween(a0, GoalPart) $>$ 10
AND angleDefinedBy(GoalPart, a0, goalie) $<$ 30
THEN levelOfQvalue(move(ahead), high)

The MLN could contain multiple formulas like this for each action. After learning weights for the formulas from source-task data, one could use this MLN to infer, given a target-task state, whether action Q-values are most likely to be high, medium, or low. Note that Q-values in RoboCup are continuous rather than discrete, so I do not actually learn rules classifying them as high, medium, or low. Instead, the algorithm discretizes the continuous Q-values into bins that serve a similar purpose.

Table 1 gives the algorithm for MLN Q-function transfer. The sections below describe the steps of this algorithm in more detail.

4.1 Learning an MLN Q-function from a Source Task

The first step of the MLN Q-function transfer algorithm in Table 1 is to divide the Q-values for an action into bins, according to the procedure in Table 2. The training example Q-values could have any arbitrary distribution, so it uses a hierarchical clustering algorithm to find good bins. Initially every training example is its own cluster, and it repeatedly joins clusters whose midpoints

Table 1. Algorithm for MLN Relational *Q*-Function Transfer.

INPUT REQUIRED
A set of batches $B = (b_1, b_2, ...)$ to consider for transfer
The *Q*-function Q^b for each batch $b \in B$
The set of games $G(b)$ that trained the *Q*-function for each batch $b \in B$
A parameter ϵ determining distance between bins
A demonstration-period length D
A validation-run length V

CREATE Q-VALUE BINS // This is a hierarchical clustering procedure.
For each batch $b \in B$
    For each source-task action a
        Determine $bins(b, a)$ for action a in batch b using Table 2
        (Provide inputs $G(b)$ and ϵ)

LEARN FORMULAS // This accomplishes MLN structure learning.
For each batch $b \in B$
    For each source-task action a
        For each $bin \in bins(b, a)$
            Let $P = \emptyset$ // These will be the positive examples.
            Let $N = \emptyset$ // These will be the negative examples.
            For each state s in a game $g \in G(b)$
                If s used action a and $Q_a^b(s)$ falls into bin
                    Set $P \leftarrow P \cup g$ // Examples that fall into the bin are positive.
                Else if s used action a and $Q_a^b(s)$ does not fall into bin
                    Set $N \leftarrow N \cup g$ // Examples that fall outside the bin are negative.
            Learn rules with Aleph to distinguish P from N
            Let $M(b, a, bin)$ be the ruleset chosen by the algorithm in Table 3
        Let $M(b, a)$ be the union of $M(b, a, bin)$ for all bins

LEARN FORMULA WEIGHTS
For each batch $b \in B$
    For each source-task action a
        Learn MLN weights $W(b, a)$ for the formulas $M(b, a)$ using Alchemy
        Define $MLN(b, a)$ as $(M(b, a), W(b, a))$
    Define $MLN(b)$ as the set of MLNs $MLN(b, a)$

CHOOSE A BATCH // Do a validation run in the source task to pick the best batch.
For each batch $b \in B$
    For V episodes: Use $MLN(b)$ as shown in Table 4 to choose actions in a new source-task run
    Let $score(b)$ be the average score in this validation run
Choose the highest-scoring $b^* \in B = argmax_b\ score(b)$

LEARN TARGET TASK
For D episodes: Perform RL but use $MLN(b^*)$ to choose actions as shown in Table 4
For remaining episodes: Perform RL normally

Table 2. Algorithm for dividing the Q-values of an action a into bins, given training data from games G and a parameter ϵ defining an acceptable distance between bins.

For each state i in a game $g \in G$ that takes 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 until too far apart.
  Join clusters c_x and c_y into c_{xy}
  $C \leftarrow C \cup c_{xy} - \{c_x, c_y\}$
  $m \leftarrow$ min distance between two new adjacent $c'_x, c'_y \in C$
Let $B = \emptyset$ // These will be the bins for action a.
For each final cluster $c \in C$ // Center one bin on each cluster.
  Let bin b have midpoint $\bar{c}$, the average of values in c
  Set the boundaries of b at adjacent midpoints or Q-value limits
  Set $B \leftarrow B \cup b$
Return B

---

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 should be 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.

The second step of the MLN Q-function transfer algorithm in Table 1 performs structure-learning for the MLN. The MLN formulas are rules that assign training examples into bins. We learn these rules using the ILP system Aleph [10]. Some examples of bins learned for *pass* in 2-on-1 BreakAway, and of rules learned for those bins, are:

IF distBetween(a0, GoalPart) $\geq$ 42
AND distBetween(a0, Teammate) $\geq$ 39
THEN pass(Teammate) has a Q-value in the interval [0, 0.11]

IF angleDefinedBy(topRightCorner, goalCenter, a0) $\leq$ 60
AND angleDefinedBy(topRightCorner, goalCenter, a0) $\geq$ 55
AND angleDefinedBy(goalLeft, a0, goalie) $\geq$ 20
AND angleDefinedBy(goalCenter, a0, goalie) $\leq$ 30
THEN pass(Teammate) has a Q-value in the interval [0.11, 0.27]

IF distBetween(Teammate, goalCenter) $\leq$ 9
AND angleDefinedBy(topRightCorner, goalCenter, a0) $\leq$ 85
THEN pass(Teammate) has a Q-value in the interval [0.27, 0.43]

From the rules generated by Aleph, our algorithm selects a final ruleset for each action. It does so using an efficient method shown in Table 3 that ap-

Table 3. Algorithm for selecting a final ruleset from a large set of rules. Rules are added to the final set if they increase the overall F measure.

Let S = rules sorted by decreasing precision on the training set
Let $T = \emptyset$ // This will be the final ruleset.
For each rule $r \in S$ // Select rules.
    Let $U = T \cup \{r\}$
    If $F(U) > F(T)$
    Then set $T \leftarrow U$
Return T

---

proximately optimizes for both precision and recall. It sorts rules from highest precision to lowest and greedily adds them to the final ruleset if they improve its F score. The combined rulesets for all the actions form the set of formulas M in the MLN.

The third step of the algorithm learns weights for the formulas using Alchemy's conjugate gradient-descent algorithm, as described in Section 3. The fourth step of the algorithm selects the best batch from among the set of candidates. We found that the results can vary widely depending on the source-task batch from which the algorithm transfers, so we use a validation set of source-task data to select a good batch.

4.2 Applying an MLN Q-function in a Target Task

The final step of the MLN Q-function transfer algorithm in Table 1 is to learn the target task with a *demonstration* approach, in which the target-task agent executes the transferred strategy for an initial period before continuing with standard RL. During the demonstration period, the target-task learner queries the MLN Q-function to determine the estimated Q-value of each action, and it takes the highest-valued action. Meanwhile, it learns normal Q-functions after each batch, and after the demonstration ends, it begins using those normal Q-functions.

The algorithm in Table 4 shows how to estimate a Q-value for an action in a new state using an MLN Q-function. For each action a, the algorithm infers the probability p_b that the Q-value falls into each bin b. It then uses 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, estimated as the average Q-value of the training data in that bin. The probability distribution that an MLN provides over the Q-value of an action could look like one of the examples in Figure 4.

Table 4. Algorithm for estimating the Q-value of action a in target-task state s using the MLN Q-function. This is a weighted sum of bin expected values, where the expected value of a bin is estimated from the training data for that bin.

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
    Collect training examples T for which Q_a falls into bin b
    Let $E[Q_a|b]$ be the average of $Q_a(t)$ for all $t \in T$
Return $Q_a(s) = \sum_b (p_b * E[Q_a|b])$

Fig. 4. Examples of probability distributions over Q-value of an action that an MLN Q-function might produce. On the left, the MLN has high confidence that the Q-value falls into a certain bin, and the action will get a high Q-value. In the center, the MLN is undecided between several neighboring bins, and the action will still get a high Q-value. On the right, there is a high likelihood of a high bin but also a non-negligible likelihood of a low bin, and the action will get a lower Q-value (this case suggests methods for intelligent exploration, which could be a direction for future work).

4.3 Experimental Results for MLN Q-function Transfer

To test MLN Q-function transfer, we learn MLNs from 2-on-1 BreakAway source tasks and transfer them to 3-on-2 and 4-on-3 BreakAway. Figure 5 shows the performance of MLN Q-function transfer in 3-on-2 and 4-on-3 BreakAway compared to standard RL and to our previous approach, macro transfer [15]. Each curve in the figure is an average of 25 runs and has points averaged over the previous 250 games to smooth over 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.

These results show that MLN Q-function transfer is comparable to macro transfer in some cases and less effective in others. In 3-on-2 BreakAway, the area under the curve for MLN Q-function transfer is not significantly different than for macro transfer ($p > 0.05$). In 4-on-3 BreakAway, the area under the curve for macro transfer is significantly higher ($p < 0.05$).

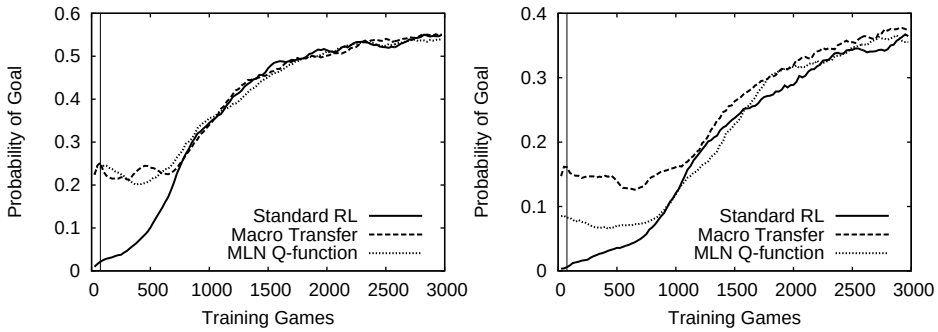

Fig. 5. Probability of scoring a goal in 3-on-2 BreakAway (left) and 4-on-3 BreakAway (right) with standard RL, macro transfer from 2-on-1 BreakAway, and MLN Q-function transfer from 2-on-1 BreakAway. The thin vertical line marks the end of the demonstration period.

5 MLN Relational Policy Transfer

MLN relational policy transfer is a method that learns an MLN to express the source-task policy, and allows target-task agents to use it for an initial demonstration period. This approach is closely related to MLN Q-function transfer, but it has the potential to transfer more effectively by focusing on policy rather than Q-values.

A policy simply determines which action to take given a state, and does not require numeric values to be assigned to actions. Thus instead of needing to create bins for continuous Q-values, MLN policy transfer learns an MLN that simply predicts the best action to take. It is also simpler than MLN Q-function transfer in that it does not need to choose a batch from which to transfer, which was a significant tuning step in the previous method.

Table 5 gives the algorithm for MLN Q-function transfer. The section below describes the steps of this algorithm in more detail.

5.1 Learning and Using an MLN Policy

The first two steps of the MLN policy-transfer algorithm in Table 5 perform structure-learning and weight-learning for the MLN. These steps are similar to those in MLN Q-function transfer. However, the formulas simply predict when an action is the best action to take, rather than predicting a Q-value bin for an action as they do in MLN Q-function transfer.

Each action may have many formulas with different weights. They are learned from examples of actions chosen in the source-task. We use only high-reward source-task episodes since those guarantee good action choices. In 2-on-1 BreakAway, these are games in which the learner scored a goal.

Table 5. Algorithm for MLN Relational Policy Transfer.

INPUT REQUIRED
Games G from the source-task learning process
A definition of high-reward and low-reward games in the source task
The demonstration-period length D

LEARN FORMULAS
Let G be the set of high-reward source-task games
For each source-task action a
    Let $P = \emptyset$ // These will be the positive examples.
    Let $N = \emptyset$ // These will be the negative examples.
    For each state s in a game $g \in G$
        If s used action a
            Set $P \leftarrow P \cup s$ // States that use the action are positive.
        Else if s used action $b \neq a$
            Set $N \leftarrow N \cup s$ // States that use a different action are negative.
    Learn rules with Aleph to distinguish P from N
    Let M be the ruleset chosen by the algorithm in Table 3

LEARN FORMULA WEIGHTS
Learn MLN weights W for the formulas M using Alchemy
Define MLN by (M, W)

LEARN TARGET TASK
For D episodes: Perform RL but choose the highest-probability action according to MLN
For remaining episodes: Perform RL normally

Some examples of rules learned for *pass* in 2-on-1 BreakAway are:

IF angleDefinedBy(topRightCorner, goalCenter, a0) ≤ 70
AND timeLeft ≥ 98
AND distBetween(a0, Teammate) ≥ 3
THEN pass(Teammate)

IF distBetween(a0, GoalPart) ≥ 36
AND distBetween(a0, Teammate) ≥ 12
AND timeLeft ≥ 91
AND angleDefinedBy(topRightCorner, goalCenter, a0) ≤ 80
THEN pass(Teammate)

IF distBetween(a0, GoalPart) ≥ 27
AND angleDefinedBy(topRightCorner, goalCenter, a0) ≤ 75
AND distBetween(a0, Teammate) ≥ 9
AND angleDefinedBy(Teammate, a0, goalie) ≥ 25
THEN pass(Teammate)

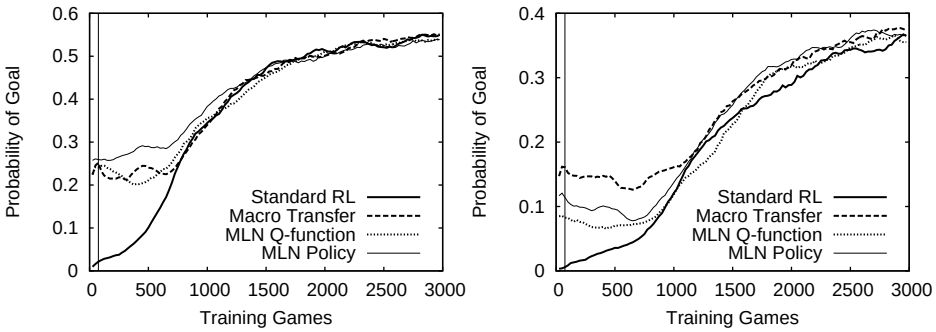

Fig. 6. Probability of scoring a goal in 3-on-2 BreakAway (left) and 4-on-3 BreakAway (right) with standard RL, macro transfer from 2-on-1 BreakAway, MLN Q-function transfer from 2-on-1 BreakAway, and MLN policy transfer from 2-on-1 BreakAway. The thin vertical line marks the end of the demonstration period.

The final step of the MLN policy-transfer algorithm in Table 5 learns the target task via demonstration. During the demonstration period, the target-task learner queries the MLN to determine the probability that each action is best, and it takes the highest-probability action. Meanwhile, it learns normal Q-functions after each batch, and after the demonstration ends, it begins using those normal Q-functions.

5.2 Experimental Results for MLN Policy Transfer

To test MLN policy transfer, we learn MLNs from the same 2-on-1 BreakAway source tasks as before and transfer them to 3-on-2 and 4-on-3 BreakAway. Figure 6 shows the performance of MLN policy transfer in 3-on-2 and 4-on-3 BreakAway compared to standard RL, macro transfer, and MLN Q-function transfer.

These results show that transferring an MLN policy is more effective than transferring an MLN Q-function, and that it can also outperform macro transfer in some cases. In 3-on-2 BreakAway, the area under the curve for MLN policy transfer is significantly higher than for both other transfer approaches ($p < 0.05$). In 4-on-3 BreakAway, it is higher than MLN Q-function transfer but still lower than macro transfer ($p < 0.05$).

6 Comparing an MLN Policy to a Ruleset Policy

The ILP rulesets that we learn for MLN policy transfer could themselves represent a policy, without the addition of an MLN. Here we perform an experiment to determine whether the MLN provides any additional benefit.

In order to use rulesets as a policy, we need a way to decide which rule to follow if multiple rules recommending different actions are satisfied in a state. To do this, we assign each rule a score. The score of a rule approximates the

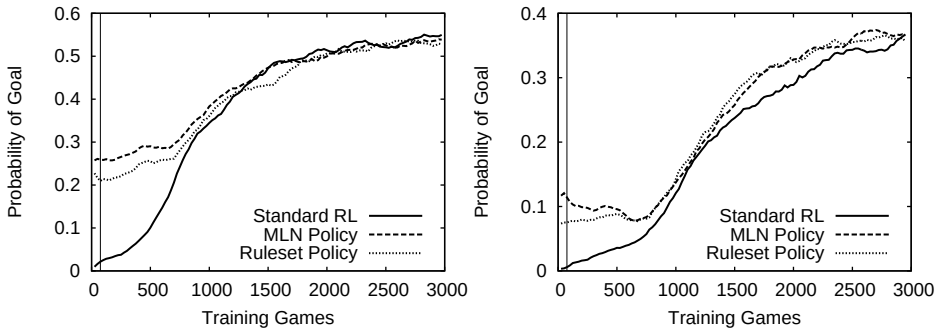

Fig. 7. Probability of scoring a goal in 3-on-2 BreakAway (left) and 4-on-3 BreakAway (right) with standard RL, regular MLN policy transfer from 2-on-1 BreakAway, and ruleset policy transfer from 2-on-1 BreakAway. The thin vertical line marks the end of the demonstration period.

probablity that following the rule will lead to a successful game, as estimated from the source-task data. At each step in the target task, we have our agents check all the rules and take the action recommended by the highest-scoring satisfied rule.

Figure 7 shows the performance of this approach in 3-on-2 and 4-on-3 BreakAway, compared with standard RL and regular MLN policy transfer. The area under the curve for rulesets is significantly less than for MLNs in 3-on-2 BreakAway ($p < 0.05$). Thus MLNs can provide an additional benefit over ILP alone. In some cases, they may be comparable; the areas are not significantly different in 4-on-3 BreakAway ($p > 0.05$).

7 MLN Policies with Action Sequence Information

MLN policy transfer assumes the Markov property, in which the action choice depends only on the current environment and is independent of previous environments and actions. However, it need not do so; the MLN formulas for action choices could use such information. Here we examine the benefit of doing so by adding predicates to the ILP hypothesis space that specify previous actions. We add predicates for one, two, and three steps back in a game. Like macros, this approach allows transfer of both relational information and multi-state reasoning.

In these experiments, Aleph only chose to use the predicate for one previous step, and never used the ones for two and three previous steps. This indicates that it is sometimes informative to know what the immediately previous action was, but beyond that point, action information is not useful.

Figure 8 shows the performance of multi-step MLN policy transfer in 3-on-2 and 4-on-3 BreakAway, compared with standard RL and regular MLN policy transfer. The area under the curve for the multi-step version is significantly

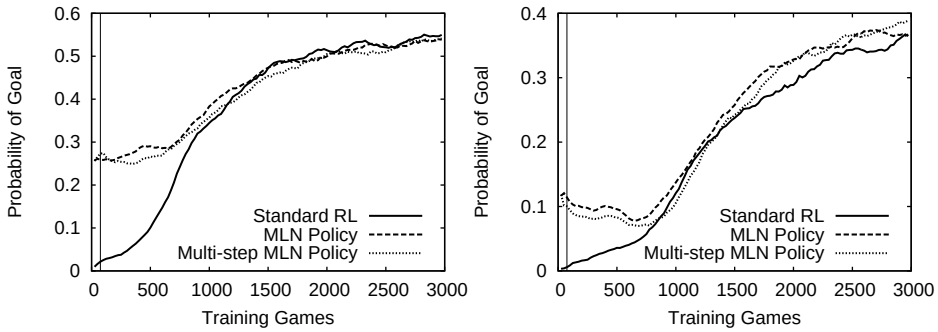

Fig. 8. Probability of scoring a goal in 3-on-2 BreakAway (left) and 4-on-3 BreakAway (right) with standard RL, regular MLN policy transfer from 2-on-1 BreakAway, and multi-step MLN policy transfer from 2-on-1 BreakAway. The thin vertical line marks the end of the demonstration period.

less than for the regular version in 3-on-2 BreakAway ($p < 0.05$) and is not significantly different in 4-on-3 BreakAway ($p > 0.05$). These results suggest that adding action-sequence information does not improve MLN policy transfer.

The Markov property appears to be a valid assumption in the BreakAway domain. While action patterns do exist in 2-on-1 BreakAway policies, and macro transfer takes advantage of them, there is apparently enough information in the current state to make action choices independently in MLN transfer. A multi-step MLN policy is therefore unnecessary in this domain, though it could be helpful in different domains where the Markov property does not hold.

8 Conclusions and Future Work

We propose algorithms for transfer in reinforcement learning via Markov Logic Networks and evaluate them in a complex domain. In MLN Q-function transfer, we represent the source-task Q-function relationally with an MLN. In MLN policy transfer, we represent the source-task policy with an MLN.

Transferring a policy with an MLN is a more natural and effective method than transferring a Q-function. Rulesets expressing a policy can be demonstrated effectively as well, but using an MLN to combine the rulesets provides additional benefits. An MLN captures complete enough information about the source task that adding knowledge about actions previously taken provides no additional benefit.

MLN policies outperform relational macros in some transfer scenarios, because they capture more detailed knowledge from the source task. However, they can perform worse in more distant transfer scenarios; in these cases they are likely capturing *too much* detail from the source task. This is a phenomenon that we call *overspecialization*, which is related to overfitting, but is specific to the context of transfer learning.

Future work in this area could focus on revision of a transferred model after the initial demonstration episodes. Our methods currently revert to standard RL, but they could instead learn by incrementally revising the source-task knowledge. Applying MLN knowledge in ways other than demonstration may also be effective.

A related area of potential work is MLN-based relational reinforcement learning. Domains like RoboCup could benefit from relational RL, which would provide substantial generalization over objects and actions. The main challenge to overcome in performing relational RL in such a complex domain is the computational cost of learning MLN structures and weights.

9 Acknowledgements

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

References

1. T. Croonenborghs, K. Driessens, and M. Bruynooghe. Learning relational skills for inductive transfer in relational reinforcement learning. In *ILP*, 2007.
2. F. Fernandez and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *AAMAS*, 2006.
3. S. Kok, P. Singla, M. Richardson, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, University of Washington, 2005.
4. D. Lowd and P. Domingos. Efficient weight learning for Markov Logic Networks. In *KDD*, 2007.
5. 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.
6. M. Madden and T. Howley. Transfer of experience between reinforcement learning environments with progressive difficulty. *AI Review*, 21:375–398, 2004.
7. 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.
8. L. De Raedt. *Logical and Relational Learning*. Springer, 2008.
9. M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.
10. A. Srinivasan. The Aleph manual, 2001.
11. P. Stone and R. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *ICML*, 2001.
12. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
13. L. Torrey and J. Shavlik. Transfer learning. In E. Soria, J. Martin, R. Magdalena, M. Martinez, and A. Serrano, editors, *Handbook of Research on Machine Learning Applications*. IGI Global, 2009.
14. L. Torrey, J. Shavlik, S. Natarajan, P. Kuppili, and T. Walker. Transfer in reinforcement learning via Markov Logic Networks. In *AAAI Workshop on Transfer Learning for Complex Tasks*, 2008.
15. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *ICML*, 2007.