Abstract
The ability to transfer knowledge from one domain to another is an important aspect of learning. Knowledge transfer increases learning efficiency by freeing the learner from duplicating past efforts. In this paper, we demonstrate how reinforcement learning agents can use relational representations to transfer knowledge across related domains.

1. Introduction
Traditional research in the machine learning community focuses on learning in the context of individual domains. For a given domain, an appropriate learning method is selected and executed. The resulting model is used within the context of the initial (training) domain, but is not typically applied toward learning in any other domain, related or otherwise.

Recent research into transfer learning takes a different view. Learners are asked to perform on a sequence of two or more related tasks. The goal is to find ways to transfer knowledge acquired in one domain (the source) into subsequent domains (the target). The desired outcome is that effort expended on learning in the source domain(s) will reduce the amount of effort required to learn in the target domain(s).

In this paper, we consider how to apply reinforcement learning to the transfer task. Toward this end, we introduce a relational version of the popular TD(λ) algorithm (Sutton, 1988). Our approach is similar to that of Dzeroski et al. (2001), who discuss the use of relational regression to learn Q-functions. This work takes a simpler approach to computing the value function, and focuses more on the underlying concept hierarchy.

2. Relational Temporal Difference Learning
Most work in reinforcement learning relies on propositional representations. This includes simple tabular approaches, the use of hand-crafted features to summarize important properties of states, and even the use of function approximators, such as backpropagation. None of these methods lend themselves to the problem of transferring knowledge from one domain to another. This is due in large part to the inflexibility of the underlying propositional representation.

We consider here a different view on approximate value function representation. Suppose that the states in a domain are factored, such that each state \( s \) is a set of relational ground literals from a finite set of possible ground literals. More formally, let \( \mathcal{L} = \{L_1, \ldots, L_n\} \) be a set of first-order predicates.

Each valid binding of constants to parameters gives a ground literal of predicate \( L_i \). Denote the set of all groundings of \( L_i \) by \( \mathcal{I}_{L_i} \). Then each state \( s_t \) is described as \( s_t = I_{L_1}(s_t) \cup I_{L_2}(s_t) \cup \cdots \cup I_{L_n}(s_t) \), in which \( I_{L_i}(s_t) \subseteq \mathcal{I}_{L_i} \) denotes the set of groundings of predicate \( L_i \) that are true in state \( s_t \). Thus, the predicates in \( \mathcal{L} \) need to provide a sufficiently complete description of a given state to distinguish its value from other states.

Given a factored state encoding, we can define a relational structure on top of this representation. Consider a hierarchical set of predicates \( \mathcal{C} = \{C_1, \ldots, C_m\} \), which we call conceptual predicates, each defined in terms of lower level predicates (including those from \( \mathcal{L} \)). The set \( \mathcal{I}_{C_i} \) then represents the groundings of \( C_i \) given a set of ground literals.

Table 1 shows sample predicates from the game of TicTacToe. The predicate Line belongs to the set \( \mathcal{L} \), which describes the game state. The predicates CanMakeLine and HasFork belong to the set \( \mathcal{C} \) and provide a more abstract view of the states. These concepts are used for evaluating states efficiently. Table 2 shows a sample result of this inference process for a particular state.
The TicTacToe domain.

Our approximate value function is distributed over this network of relational predicates \( C \). More precisely, we define a real-valued utility \( U(C) \) for every concept \( C \in \mathcal{L} \cup \mathcal{C} \) and approximate the value of any state \( s \) as

\[
V_1(s) = \sum_{C \in \text{inf}(s)} |I_C(s)| \cdot U(C) , \tag{1}
\]

in which \( \text{inf}(s) \subseteq \mathcal{L} \cup \mathcal{C} \) determines a subset of predicates that should influence the value of state \( s \).

With respect to our example, each predicate in \( \mathcal{C} \) in Table 1 (CanMakeLine and HasFork) has an associated utility value. Following inference, the value of each concept’s utility is added to the value of the state. In Table 2 (c), each member of \( I_C(s) \) adds utility to the value of a given state. Notice that, in this example, CanMakeLine has two distinct groundings. Each grounding carries the utility of the general predicate, so the value of the state shown in Table 2 (b) includes \( 2 \cdot U(\text{CanMakeLine}) + U(\text{HasFork}) \).

Combining equation (1) with the standard TD(\( \lambda \)) update rule

\[
\Delta V(s_k) = \alpha \lambda^{-k} [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]
\]

yields an update for predicate values \( U \):

\[
U(C) := U(C) + |I_C(s_k)| \cdot \Delta V(s_k) , \tag{2}
\]

for all \( C \in \text{inf}(s_k), k = 1, \ldots, t \). We call this class of learning algorithms rTD(\( \lambda \)), where “r” refers to the relational nature of the predicate hierarchy. Notice that rTD(\( \lambda \)) reduces to TD(\( \lambda \)) when there is a one-to-one mapping from states to predicates.

### 3. Experimental Demonstration

The purpose of our experimental evaluation is not to demonstrate learning in complex domains that are intractable for other learning methods. The goal is to study the behavior of rTD(\( \lambda \)) in the context of transfer learning tasks. We seek to demonstrate that rTD(\( \lambda \)) supports the representational abstractions required to transfer knowledge among related domains.

#### 3.1. The Learning Agent

The experiments reported here make use of two-player, deterministic Markov games. We therefore use a generalized version of the single-agent TD update rule (discussed above) analogous to the minimax Q-learning rule discussed by Littman (1994). This update is identical to the single agent update rule, except that each player has a separate value function computed independently from those of other players. In the alternating-turn games considered here, the minimax policy reduces to a greedy policy. The agent therefore only needs to consider its legal moves at the current state.

Our learning agent is composed of three modules. The performance module agent examines each candidate move from the current state. It then computes the approximate value of each next state using (1), and selects the action with the highest value according to the current policy. To do so, it uses the inference module to compute the predicate groundings for each candidate next state. Finally, the learning element updates predicate utilities according to (2).

### 3.2. Methodology

We begin discussion of evaluation by defining three terms. A game defines the environment in which the learner must operate. This includes the initial state, the action model, the terminal states, and the goal conditions. A match is a competition between two agents (players) in a given game (environment). Matches begin with the initial state and end when a terminal state is reached. Finally, a scenario is...
a source/target pairing of games. The agent trains in the source, and then improves performance (learning or otherwise) in the target by transferring a portion of its acquired knowledge.

To demonstrate the presence and effects of knowledge transfer, we constructed two transfer learning scenarios derived from chess. Each game description includes a set of predicates (the set $\mathcal{L}$) and a set of concepts that the agent may use to evaluate states (the set $\mathcal{C}$). Concepts for the source and target games in a scenario were identical, but the game descriptions were not. In both scenarios, the rewards were 100 for a win (agent satisfies goal), -100 for a loss (opponent satisfies goal), and 0 for a draw (neither satisfies goal).

For each scenario, we repeated the following procedure 25 times to produce a learning curve. First, we establish a non-transfer performance benchmark by training a learner for 80 matches in the target domain. Next, we train a transfer learner for 80 matches in the source game, and 80 matches in the target. After each training match in the target game, both the non-transfer and transfer learners play a test match against a fixed-policy opponent to establish a performance level. The fixed opponent was a stochastically suboptimal version of the non-transfer learner which selected suboptimal moves with probability 0.1. Note that the initial location of the pieces remains fixed for all matches.

### 3.3. Results

The first scenario (KQKQP) requires white, playing with a king and queen, to checkmate black, playing with a king, queen and pawn. Since black is ahead in the game (up by a pawn), white must find a sequence of moves which forces black’s hand. Otherwise, black will win. The target game in this scenario has the same goals as the source, but expands the powers of both queens, and adds an extra pawn for black. Both the source and target games are limited to a maximum of 14 moves before termination, producing state space sizes of approximately $10^{14}$ and $10^{15}$ respectively. Note that according to the source game description, the solution used in the source cannot be used in the target as the added pawn blocks the necessary moves.

Figure 1 shows the learning curves for both the transfer and non-transfer learning agents in the target game. The non-transfer learner requires approximately 20 self-play training matches to find the solution to the problem. The transfer learner plays correctly immediately, suggesting that no changes to its state evaluation function were required. This demonstrates that the relational representation used by $rTD(\lambda)$ is sufficiently abstract to assimilate the changes in rules between the source and target games.

Several concepts were assigned high utility by $rTD(\lambda)$ for KQKQP. One example is $queenCanBeLost/1$, which is true when a player’s queen can be captured. Also important is $kingMobility/3$, which produces one grounding for each cell that a player’s king can move to immediately, and $kingIsForced/1$ which is true if a player’s king must move (in check) and has only a single legal target cell. Less important, but still useful was $pawnQueened/1$, which is true if a player has multiple queens.

The second scenario is a new spin on an well known chess end-game. The KRK endgame requires white to checkmate black, who has a king only, using a king and rook. In this case, the source is played on a small $(5 \times 5)$ board and is limited to 14 moves, which produces a space of approximately $10^{14}$ states. The target is played on a standard $(8 \times 8)$ chess board and allows up to 18 moves, which increases the size of the state space to $10^{20}$. Although the strategies are fundamentally the same, the difference in board size changes the identity of cells critical to executing the strategy along with the number of required moves.

Figure 2 shows the learning curves for both agents. Unlike the first scenario, in which initial games were drawn and both the transfer and non-transfer learners acquired the knowledge necessary to win, neither agent learned to win regularly. However, the agent without transfer did not manage to win a single match in the target game, while the agent with transfer did ultimately improve performance and win several matches.

The most important concept in KRK is the $reachableByK-$
KRK is an example of a problem in which transfer is required for any useful learning to occur in the target. We believe that such cases represent the strongest argument in favor of research into knowledge transfer. Transfer of knowledge from one problem to another not only simplifies the learning process, as in the first scenario presented above, but also enables learning in other unfathomable domains, as in the second scenario.

4. Discussion

The work presented here demonstrates that reinforcement learning combined with a relational representation is capable of transferring knowledge from one domain to another. However, the assumption that the necessary concepts are always available to the learner is very strong. In the long term, methods that automatically construct the high-level predicates necessary to encode relational value functions are required. Our relational language provides a suitable formalization for this problem.

One possible approach for this problem is to generate relational predicates from the symbolic description of the domain, as demonstrated by Fawcett and Utgoff (1992). Statistical approaches to discovering emerging patterns have also been considered (Dong & Li, 1999). Another possibility is to use methods similar to explanation-based learning to find useful predicates from observed expert traces (Nejati et al., 2006). Confidence measures may also be useful in selecting states that require additional predicates to give more accurate value estimates.

Also important is the question of which predicates should be included in $C$. Ideally, only predicates needed to differentiate key classes of states (such as winning versus losing) would be included. Such concepts are not typically known in advance, however. In the worst case, all predicates could be included in $C$. Most predicates would be irrelevant for the purposes of state evaluation, slowing the learning process and increasing variance. Nevertheless, the presence of irrelevant concepts should not have a long-term detrimental effect on performance, provided that the agent visits a sufficient number of reward states. Notice that our experiments show that the number of training iterations required to produce useful utility values is small ($< 100$).

5. Conclusion

The $rTD(\lambda)$ framework presents a flexible method for transferring knowledge among related domains. Of particular importance is the use of a relational representation for both the state and the value function. The relational concept hierarchy provides the layers of abstraction necessary to make transfer of knowledge structures feasible. The next step toward creating a robust transfer learning system is to incorporate methods for the automatic construction of the concept hierarchies. This will improve further the flexibility of the transfer learning system.

Acknowledgments

This material is based on research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed on implied, of DARPA or the U. S. Government.

References


