Relational Temporal Difference Learning

Nima Asgharbeygi NIMAA@STANFORD.EDU
David Stracuzzi STRACUDJ@CSLI.STANFORD.EDU
Pat Langley LANGLEY@CSLI.STANFORD.EDU
Center for the Study of Language and Information, Computational Learning Laboratory,
Stanford University, Stanford, CA 94305 USA

Abstract

We introduce relational temporal difference learning as an effective approach to solving multi-agent Markov decision problems with large state spaces. Our algorithm uses temporal difference reinforcement to learn a distributed value function represented over a conceptual hierarchy of relational predicates. We present experiments using two domains from the General Game Playing repository, in which we observe that our system achieves higher learning rates than non-relational methods. We also discuss related work and directions for future research.

1. Background and Motivation

Most research in AI views intelligent behavior as search through a problem space to achieve goals. Directing that search is crucial to an agent's success, but crafting search-control heuristics manually is difficult and prone to error. An alternative response is to acquire such heuristic knowledge through learning. One common approach formulates this task as learning control policies from delayed reward, with policies encoded by expected value functions over Markov decision processes (Sutton & Barto, 1998). This general approach to reinforcement learning has been studied in many settings and from many perspectives.

Most work in this tradition uses limited representations and downplays the role of background knowledge. As a result, typical systems search a very large state space and thus learn far more slowly than do humans placed in similar situations. Research on temporal abstraction (Dietterich, 2000) and state abstraction (Asadi & Huber, 2004) aims to increase learning rates, but few efforts have utilized the more powerful relational representations that are standard in other AI subfields. Recent work on relational reinforcement learning (Dzeroski et al., 2001) uses first-order representations to provide effective abstraction, but it does not take advantage of action models, which are an important source of knowledge in many domains.

In this paper, we report a new approach to learning from delayed reward in multi-player games. Our framework is similar to relational reinforcement learning in its reliance on first-order representations. However, it employs a variant of temporal differencing, which is more appropriate than Q-learning when an action model is available, as Tesauro (1994) and Baxter et al. (1998) have demonstrated.

As in Dzeroski et al.'s work, we use a relational representation to support effective generalization across states, which should produce more rapid learning. However, rather than using relational regression trees to encode expected values, we use a factored representation that associates component values with relational predicates. These are combined into an overall score, much as in traditional state evaluation functions. Our work offers a novel approach to combining ideas from relational reinforcement learning and feature-based temporal difference learning.

In the next section, we describe our representation of states, moves, and expected values, the performance system that uses this knowledge to play games, and our method for relational temporal difference learning. We then present experiments designed to demonstrate the advantages of this approach. This includes discussion of the general game playing domain and the specific games on which we evaluate our method. We conclude by discussing related work and outlining directions for future research on relational learning from delayed reward.

Appearing in *Proceedings of the 23^{rd} International Conference on Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the authors.

2. Relational Temporal Differencing

Most work in reinforcement learning has relied on tabular representations of value functions or hand-crafted features implemented as special functions to summarize important properties of states. The former suffers from inefficiency of representation, slow learning rates, and intractability in large state spaces. Although the latter can offer compact representations and better learning rates, it is domain specific and does not provide a flexible framework to support automatic feature construction.

In this section, we propose a different view on approximate value function representation. Our framework makes a close connection to feature-based reinforcement learning, but draws on a more flexible and expressive representation of value functions over relational structures. Dzeroski et al. (2001) have pursued a similar approach that learns Q functions. This framework has an important drawback, since Q values implicitly encode both the distance to and size of the next reward (Tadepalli et al., 2004). This combination can be difficult to predict in model-free and non-deterministic environments. We focus here on the simpler case, in which an action model is available and only state values must be learned.

2.1. Reinforcement Learning in Markov Games

Consider a two-player stochastic Markov game $\langle \mathcal{S}, T, \mathcal{A}_1, \mathcal{A}_2, R_1, R_2 \rangle$, where $\mathcal{S}$ denotes the space state, $T(s'|s, a_1, a_2)$ the transition probability of reaching state s' if the players choose actions a_1 and a_2 in state s, $\mathcal{A}_i(s)$ the set of actions available to player i at state s, and $R_i(s, a_1, a_2)$ the reward that player i receives at state s in which the players select actions a_1 and a_2, respectively.

Littman (1994) generalizes single-agent Q-learning in MDPs to minimax Q-learning for zero-sum Markov games, in which we have $R_1(s, a_1, a_2) = -R_2(s, a_1, a_2)$ for all possible assignment of s, a_1, and a_2. The value functions for the two players are always negative of each other, i.e., $V_1(s) = -V_2(s)$ and $Q_1(s) = -Q_2(s)$, so only one value function must be learned. Minimax Q-learning is given by the update rule:

$$\begin{aligned} Q_1(s, a_1, a_2) := Q_1(s, a_1, a_2) + \alpha\big[R_1(s, a_1, a_2) \\ + \gamma V_1(s) - Q_1(s, a_1, a_2)\big] \ , \quad (1) \end{aligned}$$

$$V_1(s) = \max_{\pi_1 \in \Pi(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \pi(a_1) Q_1(s, a_1, a_2) \ . \quad (2)$$

Note that in a simultaneous-move Markov game the optimal policy is not necessarily deterministic. The maximum in (2) is taken over all distributions on the set of available actions $\mathcal{A}_1$. Szepesvari and Littman (1999) have shown that an agent following the minimax Q-learning algorithm converges to the optimal Q function with probability one.

Modifying minimax Q-learning to update a V function instead of a Q function is straightforward. Let λ be the parameter that determines how far back a temporal difference value should propagate. The minimax TD(λ) learning algorithm is then

$$V_1(s_k) := V_1(s_k) + \alpha\lambda^{t-k}\big[r_{1_{t+1}} + \gamma V_1(s_{t+1}) - V_1(s_t)\big] \ , \quad (3)$$

for $k = 1, \ldots, t$. This is very similar to the single-agent TD(λ) update rule, but in practice it requires more conditions to guarantee convergence. If the fixed joint policy $\pi = (\pi_1, \pi_2)$ of the two players explores all state-action pairs infinitely many times, then the algorithm will converge to V_1^π. Furthermore, if policy π converges to minimax policy in the limit of infinite exploration, then the algorithm will converge to the optimal value function V_1^*. The minimax policy $\pi = (\pi_1, \pi_2)$ is given for any state s_t by

$$\begin{aligned} \pi_1 = \arg \max_{\pi \in \Pi(\mathcal{A}_1)} \min_{a_2 \in \mathcal{A}_2} \sum_{a \in \mathcal{A}_1} \Bigg[\gamma \sum_{s} T(s|s_t, a, a_2) V_1(s) \\ + R_1(s_t, a, a_2)\Bigg] \pi(a) \ , \quad (4) \end{aligned}$$

with π_2 defined analogously for the second player.

2.2. Relational Temporal Difference Learning

Now suppose that states in $\mathcal{S}$ are factored, such that each state S is a set of relational ground literals from a finite set of possible ground literals.[1] More formally, let $\mathcal{L} = \{L_1, \ldots, L_n\}$ be a set of first-order predicates such that each L_i is an entity of the form $\texttt{Pname}(v_1, v_2, \ldots, v_k)$, with `Pname` as the name of predicate L_i and each v_i a parameter variable that can represent one of a fixed set of domain constants.

Each valid binding of constants to these variables gives a ground literal of predicate L_i. Denote the set of all groundings of predicate 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 $\mathcal{L}$ must provide a complete description of the current state. Game descriptions in the GGP framework provide an example of factored state representations.

[1] We use capital S to denote factored states, as opposed to s for non-factored states.

Table 1. A subset of predicates included in the sets $\mathcal{L}$ and $\mathcal{C}$ for the TicTacToe domain.

$\text{Line}(a_1, a_2, a_3)$
$\quad\Leftarrow\quad \text{Column}(a_1, a_2, a_3) \vee \text{Row}(a_1, a_2, a_3) \vee$
$\qquad\quad \text{Diagonal}(a_1, a_2, a_3)$

$\text{CanMakeLine}(p, b_1, b_2, b_3)$
$\quad\Leftarrow\quad (p = \text{X} \ \vee \ p = \text{O}) \wedge \text{Line}(b_1, b_2, b_3) \wedge$
$\qquad\quad \text{Cell}(p, b_1) \wedge \text{Cell}(p, b_2) \wedge \text{Cell}(\text{EmptyCell}, b_3)$

$\text{HasFork}(q, c_1, c_2, c_3, c4, c5)$
$\quad\Leftarrow\quad (q = \text{X} \ \vee \ q = \text{O}) \ \wedge \ (c_2 \neq c_4) \wedge$
$\qquad\quad \text{CanMakeLine}(q, c_1, c_2, c_3) \wedge$
$\qquad\quad \text{CanMakeLine}(q, c_1, c_4, c_5)$

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 $\mathcal{L}$). A conceptual predicate C_i can be also thought of as a function I_{C_i} that returns a set of groundings of concept C_i given a set of ground literals for each of the children predicates of C_i.

Table 1 shows sample predicates from the game of TicTacToe. The predicate *Line*, along with others such as *Cell*, *Row*, and *Column* that we assume are defined elsewhere, belong to the set $\mathcal{L}$. These predicates describe the game state. The predicates *CanMakeLine* and *HasFork*, which belong to the set $\mathcal{C}$, provide a more abstract view of the states. These concepts are used for evaluating states efficiently.

Given a state S, all true groundings of conceptual predicates can be computed by iteratively finding the ground literals of the predicates that depend only on state literals, then the parent predicates built on top of those, and so on. This produces the set of true ground literals $I_{C_i}(s)$ for every conceptual predicate C_i, i.e., the deductive closure of the predicate definitions and the state description S_t. Figure 2 shows a sample result of this inference process for a particular state. In complex domains, full deduction can be very expensive to perform at every state. Asgharbeygi and Nejati (2005) report a more efficient approach to approximate value-controlled inference.

We represent our approximate value function in a manner that is distributed over this network of relational predicates. 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

Table 2. Sample inference results (c) for TicTacToe assuming the cell constants shown in (a) and the game state shown in (b).

C_1	C_2	C_3
C_4	C_5	C_6
C_7	C_8	C_9

(a)

X		X
O	X	
O		

(b)

$I_L(S)$	$I_C(S)$
$\text{Cell}(X, C_1)$	
$\text{Column}(C_1, C_4, C_7)$	...
$\text{Row}(C_1, C_2, C_3)$	$\text{CanMakeLine}(X, C_1, C_2, C_3)$
$\text{Diagonal}(C_1, C_5, C_9)$	$\text{CanMakeLine}(X, C_1, C_5, C_9)$
$\text{Line}(C_1, C_2, C_3)$	$\text{HasFork}(X, C_1, C_2, C_3, C_5, C_9)$
$\text{Line}(C_1, C_4, C_7)$	...
...	

(c)

$$V_1(S) = \sum_{C \in W_{inf}(S)} \big| I_C(S) \big| \cdot U(C) \ , \qquad (5)$$

in which $W_{inf}(S) \subseteq \mathcal{L} \cup \mathcal{C}$ determines a subset of predicates that should influence the value of state S. This is a design decision; one might choose $W_{inf}(S) = \mathcal{L} \cup \mathcal{C}$, or restrict it to include only the highest level of conceptual predicates C_i with nonempty grounding sets $I_{C_i}(s)$.

With respect to our example, each predicate in $\mathcal{C}$ in Table 1 (*CanMakeLine* and *HasFork*) would have 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, *CanMakeRow* 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 $2 \cdot U(\mathit{CanMakeRow}) + U(\mathit{HasFork})$.

Equations (3) and (5) together let us write an update rule for predicate values U:

$$\begin{aligned} U(C) := U(C) + \alpha\lambda^{t-k} \big[& r_{1_{t+1}} + \gamma V_1(S_{t+1}) \\ & - V_1(S_t) \big] \big| I_C(S_k) \big| \ , \end{aligned} \qquad (6)$$

for all $C \in W_{inf}(S_k)$, $k = 1, \ldots, t$. We call this class of learning algorithms rTD(λ), where "r" refers to the relational nature of the predicate hierarchy. Notice that rTD is consistent with minimax TD in that it reduces to (3) when there is a one-to-one mapping from states to predicates, that is, for every state S there exists a unique predicate C that has a nonempty grounding set only at state S.

The minimax policy for rTD is still given by (4), but if we assume that players move in alternating turns, then the minimax policy reduces to greedy action selection. When the Markov game is deterministic, i.e., for any legal actions a_1 and a_2 at state S, $T(S'|S, a_1, a_2) = 1$ for exactly one $S' \in \mathcal{S}$ and zero otherwise, the policy further reduces to

$$a_1 = \arg\max_{a \in \mathcal{A}_1} \left(\gamma V_1(S') + R_1(S, a)\right) , \qquad (7)$$

when it is the first player's turn, and similarly for π_2 when it is the second player's move. Here S' denotes the next state, given that the first player performed action a in state S.

For a Markov game defined within the GGP framework, in addition to the assumptions above, the reward function $R_1(S, a_1, a_2)$ depends only on the current state S such that the total reward for each player is the sum of rewards associated with *goal* predicates that are true at the given state:

$$R_1(S) = \sum_{G \in \mathcal{G}_1} r_G \cdot 1_{\{|I_G(S)|>0\}} \ . \qquad (8)$$

Here $\mathcal{G}_1$ is a set of goal predicates for the first player which are just like other predicates except that each goal predicate $G \in \mathcal{G}_1$ has an associated real-valued reward r_G. $R_2(S)$ is defined similarly for the second player.

2.3. The Learning Agent

Our learning agent is composed of an inference engine, a performance module, and a learning element. In addition, it makes use of an external state transition simulator which, given a joint legal action $a = (a_1, a_2)$ at state S, returns the next state S' based on the GGP axioms of the game. Having the set of ground literals in current state, the inference engine deductively computes all the groundings of a given conceptual predicate hierarchy $\mathcal{C}$. Then the performance module decides what action to take at the current state.

As noted earlier, in alternating-turn games, the minimax policy in (4) reduces to a greedy one as in (7). Therefore the agent only needs to consider its legal moves at the current state (if any), perform each one mentally and update the state and predicate groundings, compute the approximate value of each next state using (5), and select the action with the highest value according to (7). After sending this action to the simulator to update the state, the learning element carries out the rTD updates on predicate utilities according to (6). In the next section, we apply rTD learning to two such games taken from the General Game Playing framework.

3. Experimental Evaluation

The purpose of our experimental evaluation is not to demonstrate learning in complex games intractable for other learning methods. Rather, the goal is to study the behavior of our relational learning method compared to non-relational approaches and to show that it has the potential for future development. More specifically, we seek to demonstrate that rTD learning exhibits superior learning rates and can take advantage of domain structure.

3.1. General Game Playing

The framework of General Game Playing (GGP) (http://games.stanford.edu) supports competition in a wide variety of N-person games. The game description language (GDL) provides a formal syntax, using a subset of first-order logic, which lets one specify states of the game and rules that players must follow (Genesereth & Love, 2005). The game manager acts as an administrator that tests moves for legality, communicates selected moves to opponents, checks for terminating conditions, and provides scores when games end.

A given GGP system inputs a logical description of a game, then uses this knowledge to play the game, including the generation of legal moves and selection among them. Unlike specialized game players, such as Deep Blue (Hsu, 2002) for chess or Chinook (Schaeffer et al., 1992) for checkers, a general game player cannot use specialized algorithms or knowledge bases of positions constructed for particular games. Rather, the system should be able to play any game that can be stated in the game description language. We believe that GGP provides an excellent setting to drive research on reinforcement learning, because it provides basic relational knowledge and offers clear performance measures on a range of game-playing tasks.

3.2. Experimental Procedure

In each game, we trained our learning agent with a series of self-play matches. This let a single agent update the concept utilities for both sides of the game. We also used an α-greedy policy during the training phase with gradually decreasing α. This ensures sufficient exploration through a greedy-in-the-limit-of-infinite-exploration policy.

Table 3. Partial list of relational predicates for TicTacToe.

$$
\begin{aligned}
&\text{IsLine}(c_1,\ c_2,\ c_3,\ m_1,\ m_2,\ m_3)\\
&\quad\Leftarrow\ \text{Cell}(x,\ y_1,\ m_1,\ c_1)\ \wedge\\
&\qquad\ \ \text{Cell}(x,\ y_2,\ m_2,\ c_2)\ \wedge\\
&\qquad\ \ \text{Cell}(x,\ y_3,\ m_3,\ c_3)
\end{aligned}
$$

$$
\begin{aligned}
&\text{M-M-B-Line}(role,\ c_1,\ c_2,\ c_3)\\
&\quad\Leftarrow\ \text{IsLine}(c_1,\ c_2,\ c_3,\ m,\ m,\ \text{B})\ \wedge\\
&\qquad\ \ \text{MarkRole}(m,\ role)
\end{aligned}
$$

$$
\begin{aligned}
&\text{CanCompleteLine}(role,\ c_1,\ c_2,\ c_3)\\
&\quad\Leftarrow\ \text{M-M-B-Line}(role,\ c_1,\ c_2,\ c_3)\ \wedge\\
&\qquad\ \ \text{Control}(role)
\end{aligned}
$$

$$
\begin{aligned}
&\text{HasFork}(role,\ c_1,\ c_2,\ c_3,\ c_4,\ c_5)\\
&\quad\Leftarrow\ \text{M-M-b-Line}(role,\ c_1,\ c_2,\ c_3)\ \wedge\\
&\qquad\ \ \text{M-M-b-Line}(role,\ c_1,\ c_4,\ c_5)\ \wedge\\
&\qquad\ \ (c2 \neq c4)
\end{aligned}
$$

$$
\begin{aligned}
&\text{CanBuildFork}(role,\ c_1,\ c_2,\ c_3,\ c_4,\ c_5)\\
&\quad\Leftarrow\ \text{M-B-B-Line}(role,\ c_1,\ c_2,\ c_3)\ \wedge\\
&\qquad\ \ \text{M-B-B-Line}(role,\ c_1,\ c_4,\ c_5)\ \wedge\\
&\qquad\ \ \text{IsLine}(c_3,\ c,\ c_5,\ \text{B},\ \text{B},\ \text{B})\ \wedge\\
&\qquad\ \ \text{Control}(role)
\end{aligned}
$$

While the system was learning, we recorded its progress after every k games by measuring its performance against a fixed-policy reference player. We define performance as the average reward that the player receives over the course of several matches. Thus, we present learning curves showing performance as a function of the number of matches the agent spent learning. For comparison purposes, we carried out the above procedure once with our relational learning system and once with a traditional non-relational TD(λ) learner with a tabular representation of value function.

During the test phase, we turned the learning off and played the agent against a reference player. We used a stochastically suboptimal player as reference, meaning that it selects a slightly suboptimal action with a probability of 0.1. This is particularly interesting because it introduces some variation into the testing procedure, as opposed to the optimal player who plays deterministically optimal.

3.3. TicTacToe

TicTacToe is played on a 3×3 board, with players putting X and O marks on empty cells with the goal of achieving a line consisting of three of their own marks. The value of the game is zero. If both players play optimally, the game will lead to a draw. We provided our system with a definition of seven relational concepts, five of which are illustrated in Table 3. This table presents predicates, stated in a Prolog-like syntax, that are part of the relational structure $\mathcal{C}$ together with their hierarchical and logical definitions that allow inference of the grounding sets $I_C(S)$.

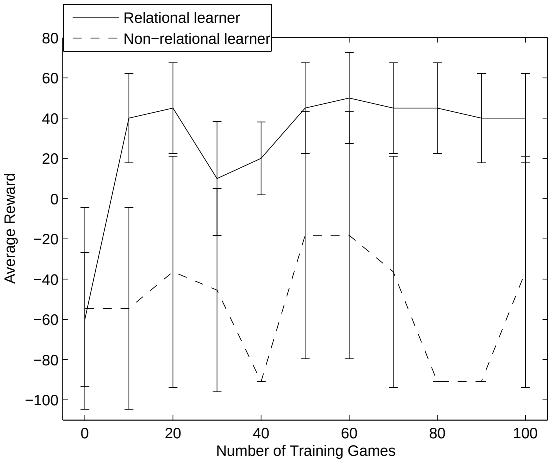

Figure 1. Performance of rTD learner playing against a stochastically suboptimal player, compared to that of the non-relational learner, at various point during learning the game of TicTacToe.

We then carried out the experiment procedure described above. We chose a typical set of parameters for our learning algorithm: α=0.1, γ=0.95, λ=0.9. We employed a two-player α-greedy policy with the probability of random actions decreasing as $50/(45 + N)$, with N being the number of games played so far. In addition, we decreased the learning parameter α in a similar manner. Our learning system was able to find the optimal policy for the game after playing only 80 self-play matches. Figure 1 shows the resulting learning curve, along with the learning curve for the non-relational version of the system in the same setting.

As expected, the learning rate for the relational system is far more rapid than that for the traditional version. The table-based learner could only show reasonable behavior after about 1000 training games, yet it was still suboptimal. This supports our hypothesis that relational knowledge structures provide generalization across different but related states.

3.4. Mini Chess

Another domain that we examined was mini Chess, which has a 4×4 board with black and white kings and one white rook. The goal for white player is to mate the black king within six moves, and the goal for black is of course to not let white succeed.

Table 4. Partial list of relational predicates for the extended version of mini Chess.

$$
\begin{array}{l}
\text{CornerCell}(c) \\
\quad \Leftarrow\ \text{Cell}(1,\ 1,\ p,\ c) \vee \text{Cell}(1,\ 5,\ p,\ c) \vee \\
\qquad \text{Cell}(5,\ 1,\ p,\ c) \vee \text{Cell}(5,\ 5,\ p,\ c)
\end{array}
$$

$$
\begin{array}{l}
\text{BlackKingTrappedOnEdge}(\text{BLACK},\ c) \\
\quad \Leftarrow\ \text{Cell}(xbk,\ ybk,\ \ \text{BK},\ c) \wedge \\
\qquad \text{Cell}(xwr,\ ywr,\ \ \text{WR}) \wedge \\
\qquad \text{EdgeCell}(c) \wedge \\
\qquad (\text{abs}(xbk - xwr) = 1 \vee \text{abs}(ybk - ywr) = 1)
\end{array}
$$

$$
\begin{array}{l}
\text{XKingsDistance}(\text{BLACK},\ x) \\
\quad \Leftarrow\ \text{Cell}(xbk,\ ybk,\ \ \text{BK}) \wedge \\
\qquad \text{Cell}(xwk,\ ywk,\ \ \text{WK}) \wedge \\
\qquad \text{Cell}(x, y, p) \wedge \\
\qquad (x \geq\ \min(xbk,\ xwk)) \wedge \\
\qquad (x <\ \max(xbk,\ xwk))
\end{array}
$$

$$
\begin{array}{l}
\text{ReachableByBlackBing}(\text{BLACK},\ x, y) \\
\quad \Leftarrow\ \text{Cell}(x,\ y,\ \ \text{B}) \wedge \\
\qquad \text{Cell}(xbk,\ ybk,\ \ \text{BK}) \wedge \\
\qquad \text{KingMove}(xbk,\ ybk,\ x,\ y) \wedge \\
\qquad (\neg\ \text{attacked}(pp,\ x,\ y)) \wedge \\
\qquad (\neg\ \text{guarded}(x, y))
\end{array}
$$

We conducted experiments following the same procedure as before. The background knowledge provided to the learning agent was composed of eight simple relational predicates, four of which appear in Table 4. The original 4×4 mini Chess was very easy for the system to learn. The learning agent found the optimal policy in less that 100 training games of self-play, so we extended the board size to 5×5 in order to have a more challenging domain. All the GGP axioms remained the same except for the predicates that define the size of the board and the initial position of the pieces on the board. We employed the same conceptual structure as for the original mini Chess.

One downside of our decision was that we no longer had access to an optimal player for the extended mini Chess. Therefore, during the test phase we took the best policy found at the end of training process, and used it to create a stochastically suboptimal reference player. Figure 2 summarizes the result of this experiment in the form of a learning curve. Once again, the learning rate appears to be promisingly high.

4. Related Research

There is a small but growing body of research on relational reinforcement learning to which the rTD algorithm is closely related. Tadepalli et al. (2004) motivate this paradigm in terms of challenges that standard reinforcement learning faces in relational domains. These include function approximations to exploit relational structure, generalization across objects, transfer across tasks, and the use of prior knowledge. Our approach to value function approximation takes direct advantage of domain structure by assigning utilities to relational predicates. This allows rTD to generalize across different ground literals for each predicate. Furthermore, our system makes explicit use of prior knowledge in learning and in value prediction.

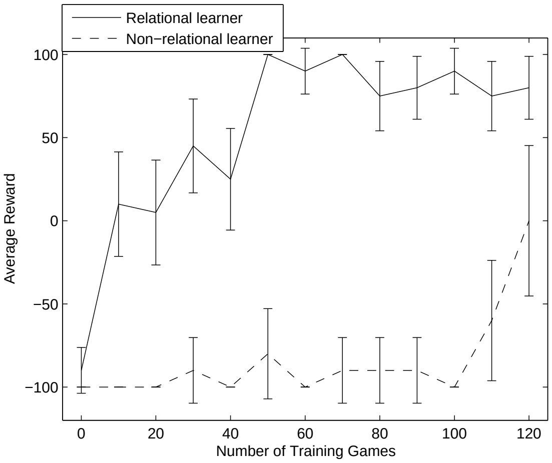

Figure 2. Learning curve for the rTD learner on the extended version of mini Chess when tested against a stochastically suboptimal player, in comparison to that of the traditional TD learner.

By contrast, previous work on relational reinforcement learning has focused on different responses to the above challenges. Dzeroski et al. (2001) adapted relational regression algorithms to take advantage of relational structure in describing Q values. For example, the TG algorithm (Driessens et al., 2001) induces a relational regression tree that predicts Q values. However, our system relies on state values so that the learning algorithm makes explicit use of the domain description.

Another closely related line of work comes from Guestrin et al. (2003), who report a system that directly approximates the value function by additively decomposing it into local functions for each class of objects, then calculating weights for the combination by solving a linear program using constraint-sampling methods. Their approach assumes that relations among objects do not change over time. Our method does not make that assumption, as our learned value function is defined over relational predicates rather than classes of objects. Less closely related approaches include approximate policy iteration (e.g., Fern et al., 2004) and explanation-based reinforcement learning (e.g., Boutilier et al., 2001).

A second area of related research comes from the application of reinforcement learning to game playing. Kaelbling et al. (1996) note that two-player games do not fit into the basic reinforcement learning framework because the goal is to maximize reward against an optimal and potentially adaptive adversary, rather than in a fixed environment. Nevertheless, Littman (1994) and others have adapted such standard algorithms to a general class of multi-player games.

One of the best known applications is Tesauro's (1994) TD-Gammon, which used temporal-difference learning to achieve professional-level play in Backgammon. More recently, Baxter et al. (1998) have reported KnightCap, a system that used temporal differencing in chess. Both relied on feature-based function approximators to estimate values, but these involved propositional features rather than relational structures. Levinson and Weber (2000) have also used learning from delayed reward to improve chess play.

5. Concluding Remarks

Our utilization of the General Game Playing framework to evaluate rTD reflects our concern with flexible approaches to learning. We believe a critical aspect of this work is the use of relational representations. Our experiments with GGP games demonstrated rapid rates of improvement and thus provide initial support for this claim. However we must augment this work with additional studies on more difficult domains, games or otherwise. We should emphasize that our work is done within the framework of temporal difference learning, and hence does not go beyond the inherent assumptions of the framework, including stationarity of learned policies.

This preliminary work is a starting point for research in two important directions. First, we should explore improvements to the value function representation and associated learning algorithms. Although we achieved impressive learning rates, we believe the representation must be enhanced to maintain this ability in more complex domains. One possible improvement would assign confidence measures to the learned predicate values so that the learning algorithm can make more informed updates toward increasing the overall confidence of state value predictions.

With respect to learning, having an effective exploration strategy is crucial, and it is well known that α-greedy exploration usually gives relatively slow convergence rates. A better alternative would be an exploration function that assigns optimistic, imaginary outcomes to under-explored states and produces greedy exploration toward unexplored regions of the state space. Confidence measures could also be useful in an exploration scheme that guides the learner toward states with less confident value estimates.

On another front, we should develop methods that automatically construct the high-level predicates necessary to encode relational value functions. Our relational language provides a suitable formalization for this problem. Several approaches hold promise, including analytical generation of relational predicates from the symbolic description of the domain (e.g., Fawcett & Utgoff, 1992), statistical approaches to discovering emerging patterns (Dong & Li, 1999), and using methods similar to explanation-based learning to find useful predicates from observed expert traces. Also, confidence measures should be useful in selecting states that require additional predicates to give more accurate value estimates.

Taken together, these extensions should produce more robust methods for reinforcement learning that take full advantage of the potential that relational representations offer. They should also let us demonstrate rapid learning over the broad range of domains that arise in the General Game Playing framework, as well as other settings in which relational structures play an important role.

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

Asadi, M., & Huber, M. (2004). State space reduction for hierarchical reinforcement learning. *Proceedings of the Seventeenth International FLAIRS Conference* (pp. 509–514). Miami Beach, FL.

Asgharbeygi, N., & Nejati, N. (2005). Guiding inference through relational reinforcement learning. *Proceedings of the Fifteenth International Conference on Inductive Logic Programming* (pp. 20–37). Bonn.

Baxter, J., Trigdell, A., & Weaver, L. (1998). Knightcap: A chess program that learns by combining TD(λ) with game-tree search. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 28–36). Madison, WI: Morgan Kaufmann.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first order MDPs. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (pp. 690–697). Seattle, Washington.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining* (pp. 43–52). San Diego, CA.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learning. *Proceedings of the Twelfth European Conference on Machine Learning* (pp. 97–108). Freiburg, Germany.

Dzeroski, S., Raedt, L. D., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Fawcett, T., & Utgoff, P. E. (1992). Automatic feature generation for problem solving systems. *Proceedings of the Ninth International Workshop on Machine Learning* (pp. 144–153). Aberdeen, Scotland.

Fern, A., Yoon, S. W., & Givan, R. (2004). Learning domain-specific control knowledge from random walks. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling* (pp. 191–199). Whistler, British Columbia.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational mdps. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (pp. 1003–1010). Acapulco, Mexico.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Levinson, R., & Weber, R. (2000). Chess neighborhoods, function combination, and reinforcement learning. *Proceedings of the Second International Conference on Computers and Games* (pp. 133–150). Hamamatsu, Japan.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 157–163). New Brunswick, NJ: Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Szepesvari, C., & Littman, M. (1999). A unified analysis of value-function-based reinforcement learning algorithms. *Neural Computation*, *11*, 2017–2060.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning* (pp. 1–9). Banff, Alberta.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program. *Neural Computation*, *6*, 215–219.