General Game Learning using Knowledge Transfer

Bikramjit Banerjee and Peter Stone
Department of Computer Sciences, The University of Texas at Austin; Austin, TX 78712
{banerjee, pstone}@cs.utexas.edu

Abstract

We present a reinforcement learning game player that can interact with a General Game Playing system and transfer knowledge learned in one game to expedite learning in many other games. We use the technique of value-function transfer where general features are extracted from the state space of a previous game and matched with the completely different state space of a new game. To capture the underlying similarity of vastly disparate state spaces arising from different games, we use a game-tree lookahead structure for features. We show that such feature-based value function transfer learns superior policies faster than a reinforcement learning agent that does not use knowledge transfer. Furthermore, knowledge transfer using lookahead features can capture opponent-specific value-functions, i.e. can exploit an opponent's weaknesses to learn faster than a reinforcement learner that uses lookahead with minimax (pessimistic) search against the same opponent.

1 Introduction

The General Game Playing (GGP) domain, introduced by Pell [1993], allows description of a wide range of games in a uniform language, called the Game Description Language (GDL) [Genesereth and Love, 2005]. The challenge is to develop a player that can compete effectively in arbitrary games presented in the GDL format. In this paper we focus on the problem of building a learning agent that can use knowledge gained from previous games to learn faster in new games in this framework.

Knowledge transfer has received significant attention recently in machine learning research [Asgharbeygi *et al.*, 2006; Taylor and Stone, 2005; Ferns *et al.*, 2006]. Instead of developing learning systems dedicated to individual applications, each beginning from scratch, inductve bias is transferred from previous learning tasks (sources) to new, but related, learning tasks (targets) in order to

- offset initial performance in the target tasks, compared to learning from scratch, and/or
- achieve superior performance faster than learning from scratch.

Oftentimes, specific skills required for target tasks are acquired from specially designed source tasks that are very similar to the targets themselves [Asgharbeygi *et al.*, 2006]. We consider the more challenging scenario where skills are more general, and source target pairs bear little resemblance to one another. Specifically, we consider the genre of 2-player, alternate move, complete information games and require that knowledge acquired from any such game be transferrable to any other game in the genre.

We develop a TD(λ) based reinforcement learner that automatically discovers structures in the game-tree, that it uses as *features*, and acquires values of these features from the learned value-function space. It then uses these values learned in one game to initialize parts of the value-function spaces in other games in the genre. The intention is to reuse portions of the value-function space that are independent of the game in our chosen genre in order to learn faster in new games. This is accomplished by focusing exploration in the complementary regions of the value function space where foresight is not informative in a game-independent way.

We use game-tree lookahead for generating features. We show that features acquired in this way against some opponent are also indicative of how to play against that opponent, even in new games. We assume that the Transfer Learner can identify whether it had played against a given opponent before (in the same or a different game) and if so, retrieve the feature values learned against that opponent for reuse. However, a simple lookahead search player would additionally need to know (or learn) the opponent's strategy to select the most effective heuristic. Without the right heuristic, we show that the lookahead search player will not perform as well as the Transfer Learner against a given opponent.

2 Reinforcement Learning

Reinforcement Learning (RL) [Sutton and Barto, 1998] is a machine learning paradigm that enables an agent to make sequential decisions in a Markovian environment, the agent's goal being to learn a decision function that optimizes its future rewards. Learning techniques emanating from RL have been successfully applied to challenging scenarios, such as game playing (particularly the champion backgammon player, TD-Gammon [Tesauro, 1994]) involving delayed rewards (rewards only on termination leading to the credit assignment problem of which actions were good/bad?). RL

problems are usually modeled as *Markov Decision Processes* or MDPs [Sutton and Barto, 1998]. An MDP is given by the tuple $\{S, A, R, T\}$, where S is the set of environmental states that an agent can be in at any given time, A is the set of actions it can choose from at any state, $R : S \times A \mapsto \Re$ is the reward function, i.e., $R(s, a)$ specifies the reward from the environment that the agent gets for executing action $a \in A$ in state $s \in S$; $T : S \times A \times S \mapsto [0, 1]$ is the state transition probability function specifying the probability of the next state in the Markov chain consequential to the agent's selection of an action in a state. The agent's goal is to learn a policy (action decision function) $\pi : S \mapsto A$ that maximizes the sum of discounted future rewards from any state s,

$$V^\pi(s) = E_T[R(s, \pi(s)) + \gamma R(s', \pi(s')) + \gamma^2 R(s'', \pi(s'')) + \ldots]$$

where $s, s', s'', \ldots$ are samplings from the distribution T following the Markov chain with policy π.

A common method for learning the value-function, V as defined above, through online interactions with the environment, is to learn an action-value function Q given by

$$Q(s, a) = R(s, a) + \max_\pi \gamma \sum_{s'} T(s, a, s') V^\pi(s') \qquad (1)$$

Q can be learned by online dynamic programming using the following update rule

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r_{sa} + \max_b \gamma Q(s', b) - Q(s, a)]$$

while playing action $a = \arg\max_b Q(s, b)$ in any state s, where $\alpha \in (0, 1]$ is the learning rate, r_{sa} is the actual environmental reward and $s' \sim T(s, a, .)$ is the actual next state resulting from the agent's choice of action a in state s. The Q-values are guaranteed to converge to those in Equation 1, in the limit of infinite exploration of each (s, a), ensured by a suitable exploration scheme [Sutton and Barto, 1998].

2.1 RL in GGP

In a General Game Playing system, the game manager acts as the environment, and a learner needs to interact with it in almost the same way as in an MDP as outlined above. Important differences are (1) the game manager returns rewards only at the end of a game (100 for a win, 50 for a draw, and 0 for a loss) with no intermediate rewards, (2) the agent's action is followed by the opponent's action which decides the next state that the agent faces. If the opponent chooses its action following a stationary (but not necessarily deterministic) policy, then the learner faces a stationary MDP as defined before. If, however, the opponent is adaptive, then the distribution T is effectively non-stationary and the above technique for value function learning is no longer guaranteed to converge. In this paper, we focus on stationary (non-adaptive) opponents.

Let $\sigma \equiv (s, a) \in \Sigma$ be the state resulting from the learner's execution of action a in state s; this is actually the state that its opponent faces for decision making. The state σ, also called an *afterstate*, can be reached from many different states of the learner as a result of different actions. So usually $|\Sigma| < |S \times A|$, and it is popular for game playing systems to learn values of afterstates, instead of state-actions. Accordingly, we learn $Q(\sigma)$.

2.2 The GGP Learner

We have developed a complete GGP learner that caters to the GGP protocol [GGP, ; Genesereth and Love, 2005]. The protocol defines a match as one instance of a game played from the start state to a terminal state, between two players that connect to the game manager (henceforth just the manager) over a network connection. Each match starts with both players receiving a GDL file specifying the description of the game they are going to play and their respective roles in the game [1] The game manager then waits for a predefined amount of time (*Startclock*) when the players are allowed to analyze the game. It is possible that both players signal that they are ready before the end of this (Startclock) phase, in which case the manager terminates this phase and proceeds with the next.

In the next phase the manager asks the first mover for its move, and waits for another while (*Playclock*). If the move is not submitted by this time, the manager selects a move at random on behalf of that player and moves to the next player, and this continues. If a move is submitted before the end of a Playclock, the manager moves to the next player early. When the manager senses a terminal state, it returns the appropriate rewards to the players, and terminates the match. In this paper we consider games where every player unambiguously knows the current state.

To measure learning performance, our learner plays a series of matches of the same game against a given opponent, and notes the cumulative average of rewards that it gets from the manager. Since the computations for transfer learning are often expensive, we perform these and the Q-updates for all states visited in the course of a match, during the Startclock of the next match. We keep the sequence of afterstates, $\sigma_1, \sigma_2, \ldots, \sigma_k$ (σ_k being terminal), in memory and use the fast TD(λ) [Sutton and Barto, 1998] update

$$\Delta Q(\sigma_p) = \alpha \lambda^{t-p}[r_{t+1} + \gamma Q(\sigma_{t+1}) - Q(\sigma_t)] \qquad (2)$$

at any time $t = 1, \ldots, k$, for $p = 1, \ldots, t$ and $\lambda \in (0, 1]$, where only r_{k+1} is potentially non-zero, with $Q(\sigma_{k+1}) \doteq 0$. Such batch update of Q-values is effectively no different from online updates since a player cannot face the same afterstate more than once in a match, unless the game allows *noop* as a move. By relegating the bulk of our computation to the Startclock period, our transfer learner makes rapid moves (involving simple Q-value lookup) which is useful in large games (such as chess derivatrives) where move-time can otherwise exceed the Playclock some times.

3 Features in Value Space

In RL, a *feature* usually means a property of some states in S. For instance, the GPS location is a feature for a mobile robot. If a set of features can be found such that the union of their joint values partitions S, then each state can be described uniquely in terms of those features. In this work we use game-specific features (or simply the state space, S) in order to enable detailed learning within each game, but for the

[1] In general, the game may not be one that the agent has ever seen before. In this paper, we consider smaller variants of four popular games, Tic-tac-toe, Othello, Connect-4 and Go, but that's mainly for ease of experimentation and presentation.

purpose of transfer, we constrain the system to identify game-independent features (the feature space) that are nonetheless correlated with the value function. These features describe the transition structure under an afterstate in the game-tree, up to a certain depth, in a *game-independent way*. For our purpose, a feature is a game tree template such that if the lookahead from a state matches the template, that feature is said to be active in that state. A feature is generated/matched by starting at the afterstate generated by one move of the learner from its current position (opponent's state shown as a red square at the root of each subtree in Figure 1) and expanding the game tree fully for up to two further moves (one move of the opponent, followed by one move of itself). The learner then classifies each node in this subtree as *win*, *loss*, *draw* or *non-terminal*. Both the tree expansion and the determination of these node classes is enabled by a game simulator (using a Prolog based theorem prover) that the learner generates from the given game description. Once all nodes in the subtree are classified, siblings of the same class in the lowermost level are coalesced. After this step, all siblings in the next higher level (i.e. the mid level in Figure 1) that have the same subtree structure under them are coalesced. The resulting structure is a feature that does not incorporate any game-specific information, such as the number of moves available to any player in any state, or the semantics of a state. Figure 1 illustrates this process. Extending this scheme to arbitrary number of lookahead levels is straightforward.

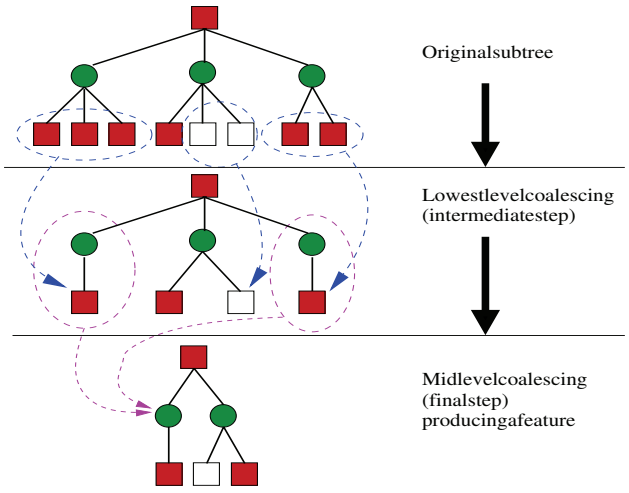

Figure 1: Illustration of an actual subtree (top) rooted at a given afterstate, matching/generating a feature (bottom). Circular (green) nodes represent the learner's states, solid (red) square) nodes are the opponent's states (or learner's afterstates). Empty squares stand for a win for the learner.

Figure 2 shows the 12 features discovered by our Transfer Learner in the Tic-tac-toe game. Note that although the features are all distinct, the associated semantics can often be overlapping; for instance, Figure 2 (j),(k) and (l) are really variants of the concept "fork opponent", since the learner's move results in a state for the opponent where no matter what move it makes, the learner can win in its next move. The Transfer Learner also needs to check if the starting afterstate

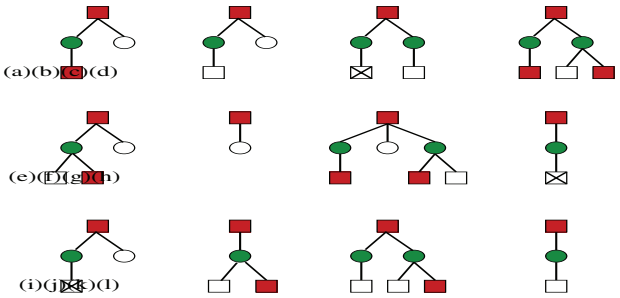

Figure 2: The 12 features discovered by the learner in Tic-tac-toe game. Empty circle/square are terminal states, with square (circle) meaning a win (loss) for the learner. A crossed square is a draw. To be considered a feature there must be at least one terminal node at some level.

is a terminal and consequently, can identify winning moves 1-step ahead.

Once the training runs are complete in the source game (in our case Tic-tac-toe), we extract feature information from the acquired value-function space. This involves matching each afterstate from the subset of Σ that was actually visited during source learning, against each of these discovered features using the simulator for lookahead. If an afterstate σ matches a feature, we note the value $Q(\sigma)$ against that feature. The value of a feature F_i is then calculated as a weighted average $val(F_i) = \text{avg}_w\{Q(\sigma)|\sigma \text{ matches } F_i\}$, where w is the weight associated with a σ, specifying the number of times σ was visited during the source game experience. Thus, the abstract features in game-tree space are associated with their values in the source task under the assumption that they will have similar values in the target task.

After the feature values have been computed, we use them to initialize $Q(\sigma)$ in the target game for each σ that matches F_i, i.e., $Q_{init}(\sigma) = val(F_i) s.t.\ \sigma$ matches F_i, once for each new σ encountered in the target game. During the Startclock of a match, we look at the afterstates visited during the preceding match. If an afterstate has not been visited in any previous match, it is matched against our set of features discovered in the source game, and initialized as above. If there is no match, we initialize to the default value [2]. Next the TD(λ) updates are done according to Equation 2.

The idea behind this transfer mechanism is to save the cost of a few value-backup steps near terminal states (i.e., when the states gain predictive potential) and thus guide exploration to focus more in the regions where foresight is not usually available. In this way, our transfer learner behaves more like human learners.

Characteristics of Feature Transfer

The features do not depend on the exact game, as long as it is within the genre of chosen games. Specifically, the size of the board, the number of available actions at each level, the semantics of states or actions, and win/loss criteria have been effectively abstracted away by exploiting the GDL. Consider the diverse natures of games in these aspects: in Tic-

[2]The default initialization value δ is the average of the win and loss rewards, in this case 50.

tac-toe the number of available moves steadily diminishes, in Connect-4 it diminishes at intervals, while in Othello it may actually increase. The winning criteria are widely varying in these games; they are similar in Tic-tac-toe and Connect-4 but completely different in Go or Othello. A key motivation behind this research is to develop simple techniques that can transfer knowledge effectively from one game to a markedly different game which is why we have focused on such a high level of abstraction.

The distinct leaf-types used in the features (Figure 2) depend on the possible outcomes of the games from which they are acquired. In this paper, we have assumed all games have 3 possible outcomes, viz., win, loss or draw, identified by distinct rewards 100, 50 and 0 respectively. If some game offers a different set of rewards (e.g., $\{-10, 0, 10, 20, 50\}$) the Transfer Learner can create a distinct leaf-type for each of these outcomes to acquire features from this game. But if it is to apply features from previous games to this game, then it needs to be provided with some equivalence relation that maps these rewards to previous reward sets, e.g., that -10 and 0 in this game corresponds to 0 in the previous games, and so on.

It is worthwhile to note that in several games such as Tic-tac-toe and Connect-4, a terminal move by any player can cause its win or a draw, but never a loss for that player. However, in other games such as Go or Othello, a player's move can cause its immediate defeat. The features discovered from Tic-tac-toe naturally cannot capture this aspect; as Figure 2 shows, there are no "win" nodes in the mid-level, or "loss" nodes in the lowest level. Our Transfer Learner can treat any of these games as source, and consequently it can capture a variety of possible types of features. In fact it can treat every game as both the application domain for previously acquired features, and at the end, as a source for new features to carry forward to future games. In this paper, however, we focus on specific source-target pairs, and learn against specific opponents to study the effects of transfer in controlled experiments.

One concern when using complex feature spaces for transfer is that the time overhead for computing transfer knowledge should not overwhelm the learning time. By having a small number of features and limiting the depth of lookahead, we are ensuring a low computational complexity for transfer knowledge. Moreover, since a single source game serves many target games, the time spent in acquiring the features is amortized, so we do not consider this as an added complexity to target learning. The limited lookahead depth also serves to keep the features somewhat indicative of the outcome of the subsequent moves. Note however, this indication is not always unambiguous, e.g., the outcome of Figure 2(g) cannot be specified without knowing the opponent's disposition. This ambiguity justifies transfer learning; if merely looking ahead would give a concrete idea of the ultimate outcome of playing a in state s irrespective of the opponent's style of play, then we could well have initialized the corresponding Q-value in the target game to the known value of that outcome, perhaps by minimax search. In the experiments, we actually show the transfer learner learning faster than RL with minimax-lookahead, against some opponents.

4 Experimental Results

In this section, we report empirical results that isolate the impact of our general game-tree-feature-based transfer scheme in a variety of games. We will consider our method to be a success if it can lead to quicker and/or better asymptotic learning in the new games when compared to learning the new games from scratch.

We extracted the feature values from the Tic-tac-toe game, the source, and tested the Transfer Learner on 3 different target games: Connect3, CaptureGo and Othello. Connect-3 is a variant of Connect-4 where the board size is 5×5 and the goal is to make a line of 3 instead of 4 pieces. CaptureGo is a variant of Go (or GoMoku) where the board size is 3×3 and a match terminates if a player captures an opponent's piece following the usual rules of Go. If no player has a move but there has been no capture yet, then the player with larger territory wins, just as in the regular version of Go. Othello follows the same rules as the regular game but is played on a smaller board of size 4×4.

For all games, we compared the learning speeds of a *baseline* learner to our Transfer Learner using feature knowledge acquired from Tic-tac-toe. The baseline learner uses afterstate TD-learning as in Equation 2 with a value function initialized uniformly to the default value. For comparison purposes and to isolate the effect of knowledge transfer from lookahead search, we also compare with a lookahead learner that uses the same depth of lookahead as the Transfer Learner, with minimax search to estimate the value of a new afterstate. In this search, non-terminal states at the leaf level are evaluated to the default value, while terminals at any level are evaluated to their actual values. The value estimate for an afterstate, thus reached, is used to initialize its Q-value for TD-learning using the same method as the other 2 learners (i.e., Equation 2).

We use three different types of opponents against which our 3 learners are made to compete in the GGP framework. These are

ϵ-greedy This opponent uses a small fixed probability, ϵ for exploration, and otherwise uses the following policy. It looks ahead one full turn and seeks terminal nodes. It takes winning moves, avoids losing moves, but otherwise plays randomly. This is similar to a shortsighted novice player.

Random This opponent picks actions using a uniform probability distribution over the set of available actions at any turn.

Weak This opponent is the opposite of an ϵ-greedy player. It explores in the same manner, but picks worst moves at decisive turns. In effect, this opponent plays randomly most of the time, but in the vicinity of a terminal state, it makes particularly poor decisions.

The purpose of considering a weak opponent is to study how fast the different learners can learn to exploit certain weaknesses in an opponent. Table 1 shows the feature values for the 12 features of Figure 2, computed by the Transfer Learner in the Tic-tac-toe game when competing against each of these 3 types of opponents. Note that the minimax-lookahead learner would initialize the afterstates that would

Table 1: Values of the features (from Figure 2) acquired in Tic-tac-toe game against various opponents.

Feature ID from Figure 2	ϵ-greedy	Random	Weak
(a)	41.08	46.31	51.62
(b)	43.75	48.81	57.55
(c)	54.36	53.75	54.63
(d)	61.5	60.11	59.63
(e)	43.03	50.41	59.62
(f)	38	43.5	40.77
(g)	40.18	49.43	58.16
(h)	50	50	50
(i)	44.64	42.98	48.93
(j)	57.13	58.9	57.48
(k)	58.42	54.84	58.28
(l)	63.36	54.26	57.45

have matched these features to the values of 0, 50 or 100. In other words the initializations of the minimax-lookahead learner are more accurate (since these are the true values for those states) than the Transfer Learner, assuming the opponent is perfectly rational. For all experiments, the learners' parameter values were $\alpha = 0.3$, $\gamma = 1.0$ (since the task is episodic), $\lambda = 0.7$, and a fixed exploration probability of $\epsilon = 0.01$.

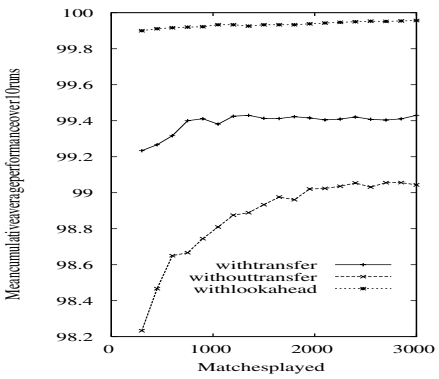

Figure 3: Learning curves for transfer learner, baseline learner, and RL with lookahead only, in 5×5 Connect3, all against ϵ-greedy opponent

Figures 3, 4 and 5 show the learning curves for the 3 learners against the ϵ-greedy opponent. The Transfer Learner uses the feature values learned against this player in Tic-tac-toe (Table 1). The cumulative average reward from last 2700 of 3000 matches are averaged over 10 runs and plotted against the number of matches in these figures. Although the Transfer Learner outperforms the baseline learner, we see that the lookahead learner is the ultimate winner since its assumption of a rational opponent is realized in this case, and it uses superior initializations compared to the Transfer Learner. Also since all the learners use fast TD methods and afterstate learning, their learning rates are high, typically crossing 95% performance level in less than 100 matches. Another thing to note from Figure 4 is that 3000 matches is insufficient for the

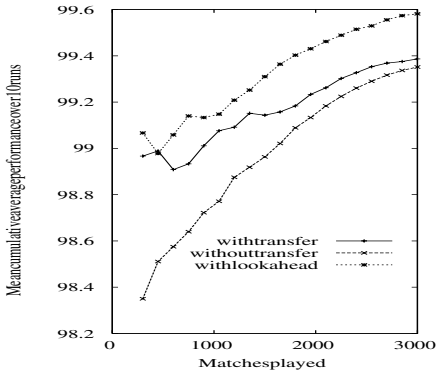

Figure 4: Learning curves for transfer learner, baseline learner, and RL with lookahead only, in 4×4 Othello, all against ϵ-greedy opponent

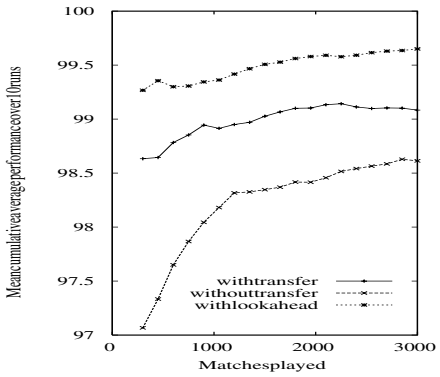

Figure 5: Learning curves for transfer learner, baseline learner, and RL with lookahead only, in 3×3 CaptureGo, all against ϵ-greedy opponent

learners to converge in the Othello game since the terminal states in this game are not as shallow as in the other games.

In order to verify the learning rates against a weak or a random opponent, we pitted the Transfer Learner and the lookahead learner against each of these opponents, in the Othello game. This game is challenging to both learners because of the depth of the terminal states. The Transfer Learner used the feature values learned against each opponent for the matches against that opponent. The learning curves are shown in Figures 6 and 7. Since the opponents are quite unlike what the minimax-lookahead learner assumes, its learning rate is poorer than the Transfer Learner. The Transfer Learner not only learns the values of features, but also learns them in the *context of an opponent*, and can reuse them whenever it is pitted against that opponent in the future. Note that the lookahead learner could have used a *maxmax* heuristic instead of minimax to learn much faster against the weak opponent, and similarly an *avgmax* heuristic against the random opponent. Because of the random policy of this opponent, the learners typically have to deal with enormous sizes of afterstate space and hence learn much slower (Figure 7) than against other opponents in the previous experiments.

These experiments demonstrate that knowledge transfer would be a beneficial addition to a baseline learner, but that if

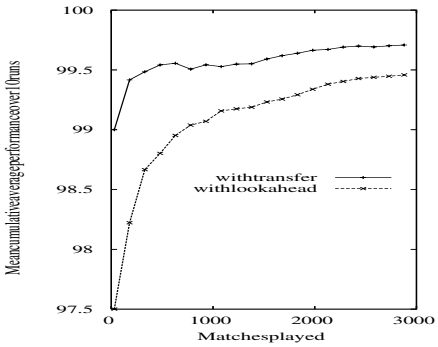

Figure 6: Transfer in 4×4 Othello against a weak opponent, compared to RL with lookahead.

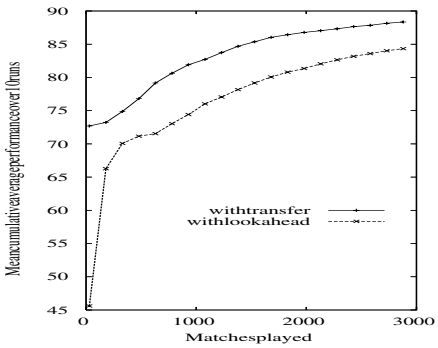

Figure 7: Transfer in 4×4 Othello against a random opponent, compared to RL with lookahead.

it implements a lookahead approach that our Transfer Learner uses as well, then its performance may be superior to the Transfer Learner, depending on the opponent. This is true if the lookahead scheme involves a heuristic that precisely matches the opponent's disposition. However, if the heuristic is a mismatch, then knowledge transfer is the better option. We argue that since selecting a heuristic (e.g., minimax) to fit an opponent is a difficult task wihtout knowing the opponent's strategy, knowledge transfer (does not need to know the opponent's strategy) is superior to lookahead learning.

5 Related Work

Lookahead search has been shown to be an effective technique in conjunction with Reinforcement Learning [Tesauro, 1994]. Automated feature discovery in games has been explored before [Fawcett, 1993], which can form the basis of further work in feature transfer. Asgharbeygi et.al. [2006] have recently developed a relational TD learning technique for knowledge transfer in the GGP domain. Their technique exploits handcrafted first order logic predicates that capture key skills in a given game and their values are learned in the same way as we do for features. The main advantage of our technique is that we do not need to define game-specific or opponent-specific features for transfer to be successful in a wide variety of games. Some of the literature on Transfer Learning for MDPs has looked into constructing correspondences between state and action spaces of two different but related MDPs [Taylor and Stone, 2005], whereas we face MDPs that have very little in common in terms of syntax or semantics of states/actions. Our approach matches the philosophy in [Ferns *et al.*, 2006] where similarity of states/actions is determined by their effects, viz. rewards and transitions through bisimulation metrics, which we accomplish by game-tree lookahead.

6 Conclusions

We have presented a Transfer Learner that uses automatic feature discovery in conjunction with reinforcement learning to transfer knowledge between vastly different 2-person, alternate move, complete information games, in the GGP framework. The key to feature construction is lookahead search of the game tree. This paper demonstrates that game-independent features can be used to transfer state value information from one game to another even better than lookahead minimax (or a fixed heuristic) search, particularly when the opponent is suboptimal. We believe the lookahead search player needs to know (or learn) the opponent's strategy, unlike the Transfer Learner, in order to select the appropriate heuristic. Even so, it is unclear whether appropriate heuristics are readily available for a variety of opponent-weaknesses (we have only studied two simple cases), and how well they work for the lookahead learner compared to the Transfer Learner. This paper shows evidence that knowledge transfer offers a simpler alternative to the complex issue of constructing appropriate heuristics for lookahead search. This work also opens up new directions in GGP and transfer learning. In the future we will extend feature definition to apply at higher levels in the game-tree, incorporate deeper features built hierarchically without deeper lookahead, and experiment with other and/or larger games.

Acknowledgements

This work was supported in part by DARPA/AFRL grant FA8750-05-2-0283 and NSF CAREER award IIS-0237699. The authors also thank Gregory Kuhlmann for providing the GGP player codebase, and Kurt Dresner for his implementation of the ϵ-greedy player.

References

[Asgharbeygi *et al.*, 2006] N. Asgharbeygi, D. Stracuzzi, and P. Langley. Relational temporal difference learning. In *Procs. ICML-06*, 2006.

[Fawcett, 1993] Tom Elliott Fawcett. Feature discovery for problem solving systems, PhD thesis, University of Massachusetts, Amherst, 1993.

[Ferns *et al.*, 2006] N. Ferns, P.S. Castro, D. Precup, and P. Panangaden. Methods for computing state similarity in markov decision processes. In *Proceedings of UAI*, 2006.

[Genesereth and Love, 2005] Michael Genesereth and Nathaniel Love. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2), 2005.

[GGP,] GGP. http://games.stanford.edu/.

[Pell, 1993] Barney Pell. Strategy generation and evaluation for meta-game playing. PhD thesis, University of Cambridge, 1993.

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

[Taylor and Stone, 2005] M.E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[Tesauro, 1994] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves masterlevel play. *Neural Computation*, 6:215–219, 1994.