Integrating Reinforcement Learning, Bidding and Genetic Algorithms

Dehu Qi
Lamar University
Computer Science Department
PO Box 10056
Beaumont, Texas, USA
dqi@cs.lamar.edu

Ron Sun
University of Missouri-Columbia
CECS Department
201 EBW
Columbia, Missouri, USA
rsun@cecs.missouri.edu

Abstract

This paper presents a GA-based multi-agent reinforcement learning bidding approach (GMARLB) for performing multi-agent reinforcement learning. GMARLB integrates reinforcement learning, bidding and genetic algorithms. The general idea of our multi-agent systems is as follows: There are a number of individual agents in a team, each agent of the team has two modules: Q module and CQ module. Each agent can select actions to be performed at each step, which are done by the Q module. While the CQ module determines at each step whether the agent should continue or relinquish control. Once an agent relinquishes its control, a new agent is selected by bidding algorithms. We applied GA-based GMARLB to the Backgammon game. The experimental results show GMARLB can achieve a superior level of performance in game-playing, outperforming PubEval, while the system uses zero built-in knowledge.

1 Introduction

How a multi-agent system can be developed in which agents cooperate with each other to collectively accomplish complex tasks is a key issue in building multi-agent systems. In this paper, we will look into a GA-based multi-agent reinforcement learning approach with bidding that learns complex tasks. We integrate three mechanisms: reinforcement learning, bidding mechanisms and genetic algorithms. That is, the learning of individual agents and the learning of cooperation among agents is completely simultaneous and thus interacting. This approach extends existing work, in that it is not limited to bidding alone. For example, not just using bidding alone to form coalitions [7] or bidding alone as the sole means for learning (as in [2]). Neither is it a model of pure reinforcement learning [6] [8]. Furthermore, it is not a pure evolutionary system [9] [4] [20]. It is the combination and the interaction of the three aspects: reinforcement learning, bidding and evolution.

2 The Algorithms and the Architecture

2.1 The GMARLB System

In our system, the multi-agent system (team) takes actions based on environment information received. Each team is composed of several member agents. Each member receives all environment information and can take action based on it. In any given state, only one member of the team is in control. The action of the whole team is chosen by the member-in-control. In the next state, the member-in-control will decide to continue to control or relinquish control. If the member-in-control decides to give up control, the new member-in-control will be chosen from all other members in that team through the bidding process. The member who has the highest bid will be the new member-in-control. In other words, the member which will more likely benefit the whole will have more chances to be chosen as the member-in-control. A snapshot of a team with 5 members is shown in Figure 1.

The member agent learns to deal with its environment through the reinforcement learning. The member-in-control receives a reward from the environment based on its actions. During the control exchange process, the current member-in-control exchanges the reward with the next member-in-control. This is also a form of communication among members.

We start our system from randomly initialized teams. After a number of episodes, we apply genetic algorithms to these teams. With the help of genetic algorithms, information not only is exchanged between members in a team, but also is exchanged between teams. The communications between members are not only through reward exchange, but also through crossover among members.

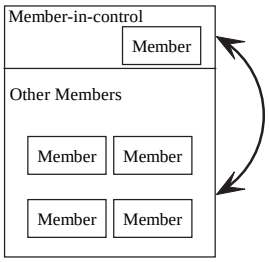

Figure 1. A snapshot of a team has 5 members. Only one member is in control in each state. The members communicate with each other through bidding.

2.1.1 The Multi-Team Algorithm

The first method is a regular genetic algorithm. We train a batch of teams as the initial population. After a number of episodes, we apply crossover and mutation to these teams. The new population is composed of the currently best teams and the newly generated teams. We call this method as the multi-team algorithm.

In the multi-team algorithm, we randomly generate a set of teams and train them for a number of training episodes. In the training process, the agent learns by playing against itself. The performance of the Q and CQ module, which will be discussed in detail in section 2.2, is improved by the reinforcement learning. In the crossover and mutation steps, teams exchange useful information to improve their performance. Only the best teams are chosen for crossover and mutation in the hope that the offspring are better than the parents. The detail of crossover and mutation operator will be discussed later. Teams are chosen by using tournament selection. The detail of the multi-team algorithm is as follows:

1. Randomly generate a set of teams. The number of teams is *n*.
2. Train each team for a number of episodes.
3. Perform crossover and mutation to generate new teams:
   (a) Select *m* best teams by using tournament selection.
   (b) Generate *n-m* new teams by crossover. The crossover rate (the percentage of the weights that have been exchanged between two members) is α.
      i. γ percent of crossover is based on the weight exchange at the corresponding position.
      ii. *100-*γ percent of crossover is based on the weight exchange at a random position.
   (c) Apply mutation on these newly generated teams by randomly mutating selected teams. The mutation rate (the percentage of the weights that have been mutated in a member) is β.
4. Replace the population with the selected teams and the newly generated teams.
5. Go to step 2.

2.1.2 Tournament Selection

For selecting the best teams, we use the tournament selection algorithm, as following:

1. Randomly divide all teams into several groups.
2. In each group, evaluate each group member's fitness value.
3. Select the best performer in each group to form a new set.
4. Repeat step 1 until *m* members are remained, where m is the number of members we needed.

In our experiments, the fitness value is the winning percentage when a member playing against the benchmark agent. For tournament selection, the higher the fitness value of a member, the higher the chance for that member to be selected. However, this algorithm is not simply selecting the best *m* members. For example, if the best two members are assigned into the same group, the second best member won't be chosen. Members ranked under *m* still have a chance to be selected.

2.1.3 The Single-Team Algorithm

In the multi-team algorithm, because the mutation and crossover are done randomly, in some cases, the performance of a newly generated team is worse than that of an old team. Therefore, we propose a new algorithm: the single-team algorithm. In the single-team algorithm, crossover and mutation are only applied in one and only one team. If the new team is worse than the old team, the new team is discarded and the old team is restored.

2.2 Details of the Reinforcement Learning with Bidding

Each member in each team is a reinforcement learning agent. For reinforcement learning, we used the Q-learning algorithm.

The Q-learning algorithm is modified for our multi-agent systems. Each member(agent) in the team (multi-agent system) has two modules: the Q module and the CQ module. In our experiments, both modules are implemented by back-propagation neural networks. Each member can select actions to be performed at each step, which is done by the Q module in the agent. For each member, there is also a controller CQ, which determines at each step whether the agent

should continue or relinquish control. Once a member relinquishes its control, to select the next agent, it conducts a bidding process among members (with regard to the current state). Based on the bids, it decides which member should take over from the current point on (as a "subcontractor"), and take the bid as its own reward.

Each member decides on its best course of actions based on the total reinforcement that it expects to receive. Each member's CQ module tries to determine whether it is more advantageous to give up or to continue, in terms of maximizing the total reinforcement that it will receive. (When it gives up, it receives a bid as its reinforcement, which represents an estimate of future reinforcement by subcontractors.) Likewise, each member (its Q module) tries to determine which action to take at each step (when it decides to continue), based on total reinforcement that it expects to receive. So, together, each member decides both types of actions based on reinforcement. Furthermore, cooperation among members is formed through the afore-described mutually sharing of reinforcement: members utilize each other when such utilization leads to higher reinforcement.

Let state s denote the actual observation by a member at a particular moment. Assume reinforcements and costs are associated with current state, $g(s)$. In each member, there are the following two modules:

- Individual action module Q: each Q module performs actions and learns through Q-learning. Each Q module tries to receive as much reward and incur as little cost as possible before it is forced to give up (including whatever it receives at the last step).
- Individual controller CQ: Each CQ module learns when the member should continue and when the member should give up. The learning is accomplished through (separate) Q-learning. Each CQ tries to determine whether it is more advantageous to terminate the member or to let it continue, in terms of maximizing its future reinforcement, which is also the overall (discounted) reinforcement.

The overall algorithm is as follows:

1. Observe the current state s.
2. The currently active Q/CQ pair (member agent) takes charge. If there is no active pair when the system first starts, go to step 5.
3. The active CQ selects and performs a control action based on $CQ(s, ca)$ for different ca. If the action chosen by CQ is end, go to step 5. Otherwise, the active Q selects and performs an action based on $Q(s, a)$ for different a.
4. The active Q and CQ perform learning based on the reinforcement received (see the learning rules later). Go to step 1.
5. The bidding process determines the next pair of Q/CQ (member) to be in control. The member that relinquished control performs learning based on the winning bid (see the learning rules later).
6. Go to step 1.

When a member gives up control, bidding goes as follows: each member submits its bid, and the member with the highest bid value wins. However, during learning, for the sake of exploration, a random selection of bids is conducted based on the Boltzmann distribution:

$$prob(k) = \frac{e^{bid_k/\tau}}{\sum_l e^{bid_l/\tau}}$$

where τ is the temperature that determines the degree of randomness in bid selection. That is, the higher a bid, the more likely the bidder will win. The winner will then subcontract from the current member and the current member takes the chosen bid as its own reward.

We dictate that the bid a member submits must be its best Q value (for the current state); in other words, each member is not free to choose its own bids. A bid is fully determined by a member's experience with regard to the current state: how much reinforcement (reward and cost) the member will accrue from this point on if it does its best. We call this an "open-book" bidding process, in which there is no possibility of intentional over-bidding or under-bidding. (However, on the other hand, due to lack of sufficient experience, a member may have a Q value that is higher or lower than the correct Q value, in which case over-bidding or under-bidding can occur). A bid submitted by a member in this way represents the expected (discounted) total reinforcement from the current point on, which is the total reward minus the total cost (including possibly its own profit as part of the cost). Note that this total represents not only what will be done by this member but also what will be done by subsequent members (subcontractors) later, due to the subsequent bidding processes (the learning process that takes this into account will be explained next). So, a member, in submitting a bid, takes into account both its own reinforcement and gains from subsequent subcontracting to other members, on the basis of its own experience thus far.

Thus, overall, the members interact and cooperate with each other through bidding as well as individual reinforcement learning. With this *dual* process, the whole multi-agent system learns to form action sequences to facilitate learning. Cooperation among members is forged through bidding and subsequent sharing of reinforcement: a member calls upon another member when such an action leads to higher reinforcement.

3 Experimental Results

3.1 Experiment Setup

One of research in artificial intelligence is programming a computer that can play board games. Board game do-

mains such as Chess [5], Check [4], GO [3], and Backgammon [17] have been popular since they have finite state spaces with well-defined rules. Since it is usually impossible to search exhaustively the state space, artificial intelligence research in game domains has primarily worked on solutions that can play a game comparable to or better than a human player.

To evaluate our multi-agent player, we play our player against two machine players: a benchmark player and Tesauro's PUBEVAL [19]. The benchmark player is a single agent, which has the same structure as the Q module in a member. The benchmark player is the best player from 15 candidates after a number of training episodes. In our experiments, we use the benchmark player to test the performance of our team players. The training time for the benchmark player is the same as that of the team player. For example, if the team player has been trained for 4,000 games, the benchmark player is the best single agent player from 15 candidates after 4,000 games training. On the other hand, PUBEVAL is a public machine player by Tesauro and it is a good evaluator for backgammon machine players. PUBEVAL uses a linear function to evaluate the board. Every board will get a point from the linear function. PUBEVAL will move the checker to the board that leads to the maximum possible point.

Our backgammon player is a GA-based multi-agent reinforcement learning team. As mentioned before, we use the back-propagation(BP) neural network to implement the Q-learning algorithm. The initial weights for BP networks are randomly generated. The BP networks trained on backgammon use an expanded scheme to encode the local information. For a player's checkers, a truncated unary encoding with five units is used to encode each checker's position (1-24, on the bar and off the board).

For encoding opponent's information, TD-Gammon's encoding scheme [17] is used. For each checker's position, a truncated unary encoding with four units is used. The first three units are encoded three cases: one checker, two checkers and three checkers, while the fourth unit encodes the number of checkers beyond 3. A total of 96 units is used to encode the information at location 1-24. In addition, 2 units are used to encode the number of opponent's checkers on the bar and off the board.

This encoding scheme thus uses 75 units for itself and 98 units for an opponent. In addition, this encoding scheme uses 12 units to encode the dice number and an additional 16 units to encode the player's first move, for a total of 201 input units.

The output encoding scheme uses 16 units to encode the checker. Among these units, 1 to 15 are the checker numbers, while 0 means no action. The hidden units of the BP network for the module Q are 40 and the hidden units of the BP network for controller CQ are 16. In the subsequent experiments, our team is composed of 5 members.

The initial parameter settings are as follows: the Q value discount rate is 0.95, the learning rate for reinforcement learning is 0.5, and the temperature is 0.50. The mutation rate is 0.05 and the crossover rate is 0.20. Eighty percentage of crossover is the weights exchanging at the corresponding position and twenty percentage of crossover is the weights exchanging at the random position.

3.2 Experimental Results

We implemented both the multi-team algorithm and the single-team algorithm. For the multi-team algorithm, 15 teams are randomly generated at the beginning. After 200 games training, 5 teams are selected for next generation. By mutation and crossover of these 5 teams, 10 new teams are generated. Plus the 5 selected systems, the new population will be trained for another 200 games.

The weights are crossed over in two ways: between the corresponding positions and between random positions. Two teams are randomly chosen and one member is chosen from each team. Sixteen percent of the weights of these two chosen members is crossed over at the corresponding position and 4 percent of the weights is crossed over at random position.

For the single-team algorithm, the team is formed by selecting the best 5 single agents after 1000 games training. The team is tested after training. If its performance is better than the old team, the crossover will be done within that team and the new team will be tested. Otherwise the old team will be restored and the old team will be crossed over again.

Similar to the multi-team algorithm, weights are crossed over in two ways: between corresponding positions and between random positions. Two members are chosen from the team. Sixteen percent of weights of these two chosen members is crossed over at the corresponding position and 4 percent of weights is crossed over at random position.

During the training, after 200 games, the team is tested by playing against the benchmark agent. The test results of the multi-team algorithm and the single-team algorithm for 4,000 games are shown in Figure 2(a) and 2(b). Both algorithms' performances against the single agent show that the GMARLB system has an overwhelming advantage over the single agent. Between these 2 algorithms, the multi-team algorithm has a better average winning percentage when compared to the single-team algorithm. And the best winning percentage for the multi-team algorithm is 92, while that of the single-team algorithm is 88. However, the training time needed for the single-team algorithm is much shorter than the multi-team algorithm. In the multi-team algorithm, we need to train 15 teams but only one team is needed in the single-team algorithm. The single-team algorithm has a

much better winning percentage/time ratio.

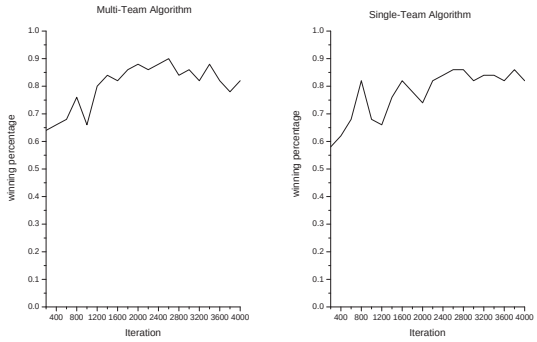

Figure 2. The winning percentage for (a)Multi-team algorithm (b)Single-team algorithm playing against the benchmark agent.

We also test our multi-team algorithm player with the benchmark agent in different game situations: full game, race game, and bearing-off game. The test results are shown in Figure 3.

In the less complicated situations (the race game and the bearing-off game), the advantage of the multi-agent system over the single agent system is not as large as that of the full game.

We continue to train multi-agent systems for the full game. The result after 400,000 games is shown in Figure 4. The highest average winning percentage is 58. The average winning percentage is the average of 5 runs, in which each run includes 50 games played against PUBEVAL every 200 iterations.

All experiments were running on HP workstations (HP-UX). The average training time for 200 games of a team is 30 minutes.

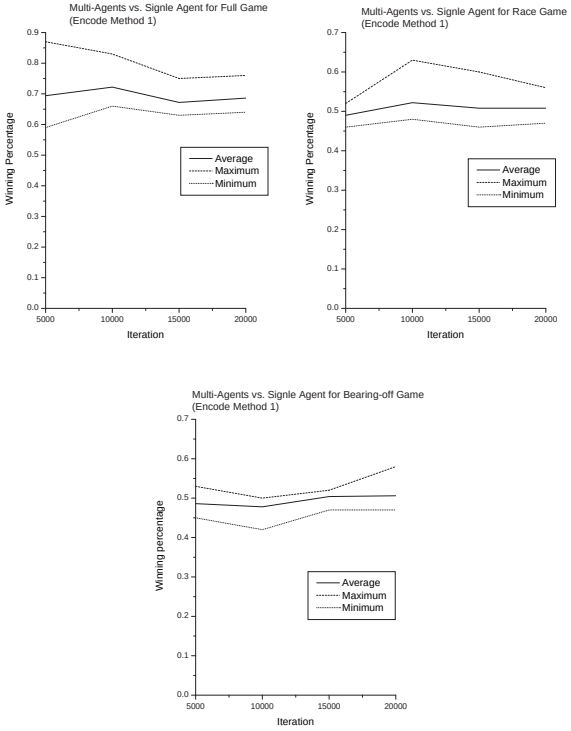

Figure 3. The winning percentage for multi-team algorithm playing against the benchmark agent in (a) Full Game (b) Race Game (c) Bearing-Off Game

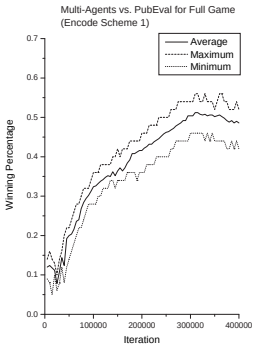

Figure 4. The winning percentage for multi-team algorithm playing against PUBEVAL in full game.

4 Analysis and Discussions

4.1 Member Analysis

We also test the performance of the team playing against its members. The results are in Table 1 and Table 2. All experiment results are for the multi-team algorithm with encoding scheme 1. The performance is measured by the winning percentage when a member played against its team in 50 games.

From the experimental results, the performance of the best team is better than the average performance of the team. For the best team, performance of the whole team may not be better than that of its best member at the beginning. But after enough training, the team beats its best

Iteration	Members in the best team play against the best team		
	Average	Best	Worst
100,000	0.456	0.62	0.38
200,000	0.46	0.52	0.36
300,000	0.456	0.5	0.38
400,000	0.444	0.48	0.40

Table 1. Members in the best team vs the best team. All data are winning percentage of a member playing against its team in 50 games.

member. While in the worst team, the performance of the worst team is not as good as its best member. We believe the reason is that the Q module in the best team is better than that of the worst team. In other words, the best team has better cooperation than the worst team.

Iteration	Members in the worst team play against the worst team		
	Average	Best	Worst
100,000	0.48	0.56	0.32
200,000	0.476	0.58	0.42
300,000	0.46	0.54	0.38
400,000	0.46	0.52	0.36

Table 2. Members in the worst team vs the worst team. All data are winning percentage of a member playing against its team in 50 games.

4.2 Comparison with Other Methods

There has been a great deal of work in the backgammon game. The best machine player so far is Tesauro's TD-Gammon [15] [16] [17]. TD-Gammon used the TD reinforcement learning algorithm [14] to learn from itself. TD-Gammon started from random initial weights but achieved a very strong level of play. Tesauro's 1992 TD-Gammon beat Sun Microsystems' Gammontool and his own Neurogammon 1.0, which trained on expert knowledge. His 1995 player incorporated a number of hand-crafted expert-knowledge features, including concepts like existence of a prime, probability of blots being hit, and probability of escaping from behind the opponent's barrier. This new player achieved the world master player level. Since the best player by Tesauro is not public, we can only use his PUBEVAL, which is close to the best, to evaluate our player.

Pollack et al [10] [11] [18] used feed-forward neural network to develop a backgammon player called HC-Gammon. The neural network does not have an error back-propagation learning part. The player is first generated by random weights and then the network is mutated. The mutated player plays a few games against the original player. If the mutated player wins more than half of games, it survives for next generation. The best player generated by this method wins about 45 percent of games when played against PUBEVAL.

Sanner [12] used the ACT-R theory of cognition [1] to train a backgammon player called ACT-R-Gammon. ACT-R is an empirically derived cognitive architecture intended to model the data from a wide range of cognitive science experiments. ACT-R-Gammon achieved 40 winning percentage when played against PUBEVAL. However, since it used hand-coding of high-level function to facilitate the learning, we do not include this method in our comparison table.

The comparison with other backgammon players is in Table 3.

	Our Player	TD-BackGammon	HC-Gammon
Winning Percentage	51.2 [56]	59.25	40 [45]
Iteration	400,000 games	More than 1,000,000 games	100,000 generations

Table 3. Comparisons with other backgammon players. The number is the average of winning percentage of some runs when played against PUBEVAL. The number in brackets is the highest winning percentage.

Among those backgammon players, HC-Gammon has the shortest time to reach 40 winning percentage when playing against PUBEVAL. Our players reach 50 winning percentage in a shorter time. TD-gammon has a slighter better winning percentage. However, it has hand-crafted heuristic codes to improve performance.

4.3 Discussions

Cooperation in multi-agent systems. Backgammon is a game based on random numbers (dice numbers), so it is impossible to use the look ahead method as is usually done in other games. Rather than relying on the ability to look ahead, to work out the future consequences of the

current state, players of backgammon rely more on judgement to accurately estimate the value of the current board state, without calculating the future outcome. That makes the backgammon unusual in comparison with the other games, such as chess and GO. Although our program did not achieve the same level as TD-Gammon, it achieves a superior level with much less training time and starting from scratch. The cooperation among agents (through bidding) helps to simplify the learning process.

Although the best team does not necessarily includes all the best members in the population, it beats other teams because it has better cooperation among agents. We believe CQ modules are very important for our multi-agent systems.

Hierarchical reinforcement learning. This work also makes some interesting connections to hierarchical reinforcement learning [13]. Our approach amounts to building three-level hierarchies automatically through agent competition (bidding), without relying on extra knowledge or assumptions about domains. The lower layer is a neural network, the middle layer is reinforcement learning with bidding, and the upper layer is the genetic algorithm. All 3 components in our system, GA, RL, and bidding, are important. Missing any component leads to poorer performance.

Co-evolution. The agent in this learning system has two roles: teacher and student. The teacher's goal is to correct the student's mistakes, while the student's goal is to satisfy the teacher and avoid correction. Each agent can be either teacher or student, which depends on its performance in the current stage. The self-learning and self-teaching among agents help our backgammon player to achieves a superior level with zero built-in knowledge.

5 Conclusions

In sum, in this work, we developed a GA based bidding approach for performing multi-agent reinforcement learning, to form action sequences to deal with a complex situation: the backgammon game. The experimental results show the advantage of the bidding system over the single reinforcement learning, the pure GA approach, and the bidding reinforcement learning system without GA. The experiment shows the GA-based bidding system can achieve a superior level of performance in game-playing programs while the system uses zero built-in knowledge. The result of the experiments suggests that the bidding system may work well in general complex problems.

References

[1] J. R. Anderson and C. Lebiere. *The atomic components of thought*. Lawrence Elbaum Associates, Mahwah, NJ, 1998.

[2] E. B. Baum. Manifesto for an evolutionary economics of intelligence. *Neural Networks and Machine Learning*, pages 285–344, 1998.

[3] B. Bouzy and T. Cazenave. Computer go: An ai oriented survey. *Artificial Intelligence*, 132:39–103, 2001.

[4] D. B. Fogel. Evolving a checkers player without relying on human expertise. *Intelligence, ACM Press*, (Summer):21–27, 2000.

[5] F. Hsu, T. Anantharaman, M. Cambell, and A. Newatyzk. A grandmaster chess machine. *Scientific American*, 263(4):44–50, 1990.

[6] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, Madison, WI, 1998.

[7] S. Ketchpel. Forming coalitions in the face of uncertain rewards. In *Proceedings of AAAI*, 1994.

[8] P. Maes, R. H. Guttman, and A. G. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, 1999.

[9] A. G. Moukas and G. Zacharia. Evolving a multiagent information filtering solution in amalthaea. pages 394–403, Marina del Rey, CA, 1997. ACM Press.

[10] J. Pollack and A. Blair. Coevolution of a backgammon player. In *Proceedings of the Fifth Artificial Life Conference*. MIT Press, 1996.

[11] J. Pollack and A. Blair. Co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:225–240, 1998.

[12] S. Sanner, J. Anderson, C. Lebiere, and M. Lovett. Achieving efficient and cognitively plausible learning in backgammon. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 823–830. Morgan Kaufmann, 2000.

[13] R. Sun and T. Peterson. Multi-agent reinforcement learning: Weighting and partitioning. *Neural Networks*, 12(4-5):127–153, 1999.

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

[15] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[16] G. Tesauro. Td-gammon, a self teaching backgammon program, achieves master-level play. *Neural Computing*, 6(2):215–219, 1994.

[17] G. Tesauro. Temporal difference learning and td-gammon. *Communications of ACM*, 38(3):58–67, 1995.

[18] G. Tesauro. Comments on 'co-evolution in the successful learning of backgammon strategy.'. *Machine Learning*, 32:241–243, 1998.

[19] G. Tesauro and T. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390, 1989.

[20] X. Yao and Y. Liu. From evolving a single neural network to evolving neural network ensembles. In M. J. Patel, V. Honavar, and K. Balakrishnan, editors, *Advances in the Evolutionary Synthesis of Intelligent Agents*, pages 383–428. MIT Press, Cambridge, MA, 2001.