International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.1, No.1 (2005), pp. 42–58

MARLBS: Team Cooperation through Bidding

Ron Sun[1] **and Dehu Qi**[2]

[1]Cognitive Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
rsun@rpi.edu

[2]Department of Computer Science
Lamar University
Beaumont, TX 77710
dqi@cs.lamar.edu

Abstract: **A cooperative team of agents may perform many tasks better than isolated agents. The question is how cooperation among self-interested agents may be achieved. It is important that, while we encourage cooperation among agents to form a team, we maintain autonomy of individual agents as much as possible, so as to maintain flexibility and generality. This paper presents an approach toward this goal, based on bidding utilizing reinforcement values acquired through reinforcement learning. The result is a simple and straightforward method that is generic and works in a variety of task domains. We further apply evolutionary computation to enhance cooperation among agents of a team, through selecting and reproducing those teams that are able to cooperate. We tested and analyzed this approach, MARLBS, in a variety of task domains, and demonstrated that a team of self-interested agents indeed performed better than the best single agent as well as the average of the single agents. In particular, Backgammon players trained using this approach outperformed PubEval (a publicly available benchmark player). These results validated our approach.**

I. Introduction

Multi-agent systems (MAS) have become an important focus of artificial intelligence, and useful in many emerging application areas of AI. However, there are many significant research issues concerning the coordination and cooperation of multiple agents. These issues are somewhat analogous to issues studied in distributed systems (including distributed AI), such as those concerning task decomposition and solution synthesis. In distributed systems, typically, subproblems are not completely independent, but they may be separately solved to obtain sub-solutions. Then these subsolutions can be synthesized into a solution of the original problem. In multi-agent systems, different from typical distributed systems, such "decomposition" and "synthesis" are not centralized, but accomplished through emergent means (Castelfranchi 2001), which makes the issues more difficult.
Increasingly, machine learning is being used as a vital component of multi-agent systems. Solutions to many tasks are envisioned with teams of agents (e.g., robots) learning to cooperate to achieve global objectives. In addition, some machine learning techniques (such as evolutionary computation) may be applied to designing large populations of self-interested agents that interact with each other to solve problems.
Multi-agent learning, however, poses significant theoretical challenges, particularly in understanding how agents can learn and adapt in the presence of other agents that are simultaneously learning and adapting. Fairly recently, there have been significant theoretical developments relevant to learning in MAS, in fields such as Bayesian, game-theoretic, decision-theoretic, evolutionary, and reinforcement learning. Co-learning involving multiple agents has been studied in a few different disciplines under various guises for quite some time. For example, the issue has been addressed by distributed artificial intelligence, parallel distributed computing, social psychology, game theory and other areas of mathematical economics, sociology, anthropology, and many other related disciplines. These approaches and techniques need to be tested in more challenging multi-agent applications.
In multi-agent settings, a key theoretical issue is: How can a multi-agent system be developed in which agents cooperate with each other to collectively accomplish complex tasks without being externally dictated to do so? For example, the game theoretic notion of a "coalition" has been applied in this area (Kahan and Rapoport 1984). Although game theory provides analyses of variously defined states of equilibria in coalition formation that may be achieved by self-interested agents pursuing their own interests, it does not study sufficiently how these equilibria can be achieved computationally

(e.g., through learning). It usually makes the assumption of completely rational agents that can examine all the aspects of a coalition (although limited rationality models have been looked into). The problem of optimal coalition formation could be NP-complete (Kahan and Rapoport 1984). There are also significant philosophical issues concerning the relationship between individual self-interested actions and the collective function that they achieve, which is yet to be fully understood (Sun 2001, Castelfranchi 2001). Therefore, there is a need for more experimental work in model building and idea testing that can shed additional light on the issues, as they may lead to better theoretical models in the future as well as more successful applications.

We are interested in the following research issues relevant to multi-agent learning: (1) how self-interested agents learn to cooperate with each others (under bounded rationality); (2) what a minimum mechanism should be for establishing cooperation among self-interested agents; (3) what the role of social interaction, such as bidding and negotiation, is in co-learning; (4) how we can integrate individual learning and collective evolution in fostering cooperation.

The goal of the present paper is to develop a general-purpose multi-agent learning model. Although we do not yet have rigorously derived theoretical answers to these above questions (which, we believe, will not be adequately achieved for a long time to come), we present here our learning model that represents our attempt at experimentally addressing some of these issues. This learning model incorporates reinforcement learning, bidding algorithms, and genetic algorithms.

The model, named MARLBS (standing for *the Multi-Agent Reinforcement Learning Bidding System*), deals with a special case of forming a "coalition" (or a "team"), without the high cost of forming game theoretical coalitions. In the process, we mix two mechanisms, reinforcement learning and bidding, completely. That is, the learning of individual self-interested agents and the learning of cooperation among these self-interested agents are simultaneous and thus interacting. This model extends existing work, in that it is not limited to bidding alone, for example, not just bidding alone for forming coalitions (as in Rosenschein and Zlotkin 1994) or bidding alone as the sole means for learning (as in Baum and Durdanovic 2000). Neither is it a model of pure reinforcement learning, without explicit interaction among agents (such as Shoham and Tennenholtz 1994, Hu and Wellman 1998, Littman 2001). It addresses the combination and the interaction of the two aspects: reinforcement learning and bidding. On top of that, evolution is used for further enhancing cooperation among agents. Learning and evolution are also interacting in this multi-agent model.

It is important to note that here we are talking about forming a coalition (or team) of self-interested agents. These agents do not start out with a mutual commitment to teamwork. Rather, their cooperation is formed through a learning process and out of their self interest. They are not dictated to do so. We do not rely on BDI frameworks to capture team cooperation. In other words, this work is about learning to form teams, rather than usual pre-determined team models.

II. The Model

A. Basic Ideas

In a multi-agent system, an agent is an autonomous and self-interested entity that has the power to act on its own initiative. The environment usually contains other agents, which is one of the reasons why the world is dynamic. A multi-agent system is a dynamic system in which the performance and the learning of an agent depends on the actions (and the learning) of other agents. Agents in a multi-agent system may (or may not) communicate with each other (explicitly or implicitly).

We developed a multi-agent learning model MARLBS (Sun and Sessions 1999, 2000; Qi and Sun 2003). In this model, a team is composed of a number of member agents. Each member receives environmental information (full or partial observations) and can make action decisions based on it. In any given state, only one member of the team is in control. The team's action is the active (controlling) member's action. Each team member learns how to take actions in its environment through reinforcement learning when it is active (i.e., when it is in control of the actions of the team).

In each state, the member in control decides whether to continue to be in control or relinquish control. If it decides to relinquish control, another member will be chosen to be in control through a bidding process. That is, once the current member in control relinquishes its control, to select the next agent, it conducts a bidding process among members of the team (who are required to bid their best Q values; more later). Based on the bids, it decides which member should take over next from the current point on (as a "subcontractor"). Generally speaking, the member who submits the highest bid will likely be chosen as the new member in control. The current member in control then takes the bid as its own payoff. Thus, the member who is more likely to benefit from being in control, and to benefit the other agents on its team, is likely to be chosen as the new member in control. A snapshot of a team with 5 member agents is shown in Figure 1.

The member in control receives payoffs from the environment based on its actions. This constitutes a form of communication between an agent and its environment. During the control exchange process (with bidding), the current member in control also receives payoffs from the next one (in the amount of the accepted bid). This is, in fact, an implicit form of communication among members. Since all members of a team have the same structure and the same ability to receive environmental information, a team is a homogeneous communicating system. Since bidding is used to distribute control among team members, distribution of control is competitive in this system.

Each member agent of the team has two modules: the Q module and the CQ module. Each member agent may select an action to be performed at a step, which is done by its Q mod-

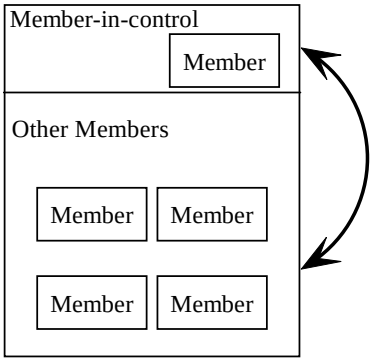

Figure. 1: A snapshot of a team with 5 members. Only one member is active (i.e., in control) at each step. The members communicate with each other through bidding.

ule. The controller module CQ determines whether the agent should continue or relinquish its control whenever the agent is in control. The structure of a member agent is shown in Figure 2.

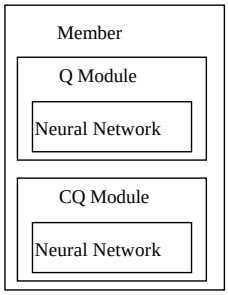

Figure. 2: The structure of a team member (an agent) in MARLBS.

The Q-learning algorithm of Watkins (1989) is modified for our multi-agent teams (cf. Claus and Boutilier 1998, Sun and Qi 2000). Both the Q module and the CQ module learn through Q-learning. The values resulting from Q-learning are used not only for guiding the actions of an agent, but also for bidding (see details later).

Evolution is used to enhance multi-agent teams via a process of team generation and competition. It produces successive generations by repeatedly mutating and recombining some of the best teams of the current generation. The best teams are those that are capable of cooperation through bidding (more discussions of this point later). That is, genetic algorithms are applied for the co-evolution of a team of agents: There is selective pressure for agents to cooperate. Agents in a team must learn to cooperate in order to survive the selection process. Thus, evolution helps to form multi-agent cooperative teams. In addition, as agents *learn* to cooperate with each other, the results of learning affect evolution. Thus, there is a strong coupling between learning and evolution.

B. Model Details

Let us look into details. Let state s denote the observation (full or partial) by a team at a particular moment. We assume reinforcement (payoffs and costs) is associated with the current state, denoted as $g(s)$.

Agent structures. In each member agent, there are the following two modules:

- Individual action module Q: Each Q module selects and performs actions, and each learns through Q-learning (Watkins 1989). Each Q module tries to receive as much payoff (and incur as little cost) as possible before it is forced to give up (including the payoff it receives at the last step, in the form of the expected value of the accepted bid; more later).
- Individual controller module CQ: Each CQ module learns when the agent should continue and when it should relinquish control. The learning is accomplished through Q-learning (separate from that of the Q module). Each CQ tries to determine whether it is more advantageous to terminate the control by the agent or to let it continue, in terms of maximizing its total reinforcement. [1]

The overall algorithm for a team of agents is as follows:

1. Observe the current state s.
2. The current active agent on the team (the current agent in control) takes control. If there is no active agent (when the team first starts), go to step 5.
3. The CQ module of the active agent selects and performs a control action based on $CQ(s, ca)$ for different ca (i.e., "continue" or "end"). If the action chosen by CQ is end, go to step 5. Otherwise, the Q module of the active agent selects and performs an action based on $Q(s, a)$ for different a.
4. The active agent (both of its Q and CQ modules) performs learning based on the reinforcement received (see the learning rules later). Go to step 1.
5. The bidding process (which is based on current Q values of agents) determines the next member agent to be in

[1] Technically, we can merge Q and CQ. But we keep them separate, for the following reasons: (1) to make function approximators simpler, (2) to make the scheme conceptually clearer, (3) to address possible extensions to more than two levels (Sun and Sessions 2000), or to using different input representations at different levels (even in the case of two levels).

control. The agent that relinquished control performs learning taking into consideration the expected winning bid (which is its payoff for giving up control; see the learning rules later).

6. Go to step 1.

Bidding is conducted as follows: Each member of the team submits its bid, and the one with the highest value wins. However, during learning, for the sake of exploration, a stochastic 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. The higher a bid is, the more likely the bidder will win. The winner will then subcontract from the current member in control. The current member in control takes the chosen bid as its own payoff. [2]

As mentioned before, one crucial aspect of this bidding process is that the bid a team member submits must be its best Q value for the current state; in other words, each member agent is not free to choose its own bids. A Q value resulting from Q-learning (see Watkins 1989 and the discussion of learning rules later) represents the total (discounted) reinforcement that an agent may receive based on its own experience. Thus, a bid is fully determined by a member's experience with regard to the current state: how much reinforcement (payoff and cost) the member will accrue from this point on, if it does its best. Moreover, an agent, in submitting a bid (the best Q value for the current state), takes into account both its own gains and the gains from subsequent subcontracting to other agents (because it takes the accepted bid as its own payoff when subcontracting).

We call this an "open-book" bidding process, in which there is no possibility of intentional over-bidding or under-bidding. (However, due to lack of sufficient experience, an agent may have a Q value that is higher or lower than the correct Q value, in which case unintentional over-bidding or under-bidding can occur).

The learning rules. The Q-learning algorithm of Watkins (1989) is modified for team reinforcement learning, for the sake of establishing cooperation among agents (cf. Claus and Boutilier 1998, Sun and Qi 2000). The learning rules may be described as follows:

- For the active Q_k, the learning rule when neither the current action nor the next action by the corresponding CQ_k is end is the usual Q-learning rule:

$$\Delta Q_k(s,a) = \alpha(g(s) + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a))$$

where s' is the new state resulting from action a in state s, a' is any action in the new state s', $g(s)$ is the reinforcement for the current state s, α is the learning rate, and γ is the discount rate (which prefers reinforcement received sooner rather than later). Thus, the agent accumulates reinforcement along the way. The Q value of a state-action pair represents the expected (discounted) total reinforcement that the agent will receive from that point on (Watkins 1989). [3]

When the next action by CQ_k is *end*, the Q_k module receives as payoff the value of CQ_k (which represents the expected value of the chosen bid at this point):

$$\Delta Q_k(s,a) = \alpha(g(s) + \gamma CQ_k(s', end) - Q_k(s,a))$$

where s' is the new state (resulting from action a in state s) in which control is relinquished by CQ_k. $CQ_k(s', end)$ represents the expected value of the bid that the agent will accept (from subcontractors), if it gives up control at this point. This value is given to the Q module so that it can take this payoff into account when deciding on its course of action (e.g., whether to reach one giving-up point or another). [4]

Therefore, summarizing the above two learning rules, a Q value of an agent (for a particular state-action pair) is the expected (discounted) total reinforcement that the agent will receive from that point on. The Q module of an agent then decides the actions of the agent based on maximizing the expected (discounted) total reinforcement that the agent will receive.

- For the corresponding CQ_k, there are also two separate learning rules, for the two different actions. When the current action by CQ_k is $continue$, the learning rule is the usual Q-learning rule:

$$\Delta CQ_k(s, continue) =$$

$$\alpha(g(s) + \gamma \max_{ca'} CQ_k(s', ca') - CQ_k(s, continue))$$

where s' is the new state resulting from action $continue$ in state s and ca' is any control action by the CQ (i.e., "continue" or "end"). That is , when the CQ_k decides to continue, it accumulates reinforcement generated by the actions of the corresponding Q_k.

When the current action by the CQ_k is end, the learning rule is:

$$\Delta CQ_k(s, end) = \alpha(\max_a Q_l(s,a) - CQ_k(s, end))$$

where Q_l denotes the Q value of the next member in control (the chosen bidder l) and $\max_a Q_l(s,a)$ is the

[2] We do not allow the current member in control who decided to relinquish control to participate in bidding itself, so as to avoid unnecessary calls for bidding.

[3] Only the member in control performs learning at each step. Other agents do not. In this way, agents become specialized.

[4] Alternatively, we may use directly the winning bid here. Using the expected bid value tends to speed up learning.

chosen bid. That is, when a CQ ends, it takes the chosen bid as its payoff, which gives it incentive to take higher bids. It learns the expected value of bids through this rule.

So, summarizing the above two learning rules, in effect, the CQ module makes its continue/end decisions based on comparing whether relinquishing control or continuing control will lead to higher expected (discounted) total reinforcement from the current point on. Agents are rational in this regard.

As we can see from the learning rules above, a Q value of an agent includes all the reinforcement it expects to receive, including the bid from the next agent. The bid submitted by the next agent is its (best) Q value, so it includes all the reinforcement the next agent expects to receive, including the bid by the third agent following it. The bid submitted by the third agent includes all the reinforcement the third agent expects to receive, including the bid by the fourth agent. This process goes on. Eventually, a Q value of an agent includes all the reinforcement expected to be received by the agent itself and by all the subsequent agents (subcontractors). Similarly, a CQ value also represents all the reinforcement expected to be received by the agent itself and by all the subsequent agents (subcontractors). In an agent, both the Q module and the CQ module decide their actions based on the total (discounted) reinforcement achieved from the current point on by the team, not just by the original agent itself. Therefore, agents of a team are incentivized to cooperate with each other, in order to achieve higher total reinforcement. See Sun and Sessions (2000) for more detailed analyses of these reinforcement learning rules.

In any multi-agent systems, each agent participates in some way to accomplish a task. When an episode (e.g., a game) is over, each member is assigned a portion of the total payoff. This raises an issue: How do we properly and fairly assign payoffs to each team member? This is the well known credit assignment problem. Bidding resolves the credit assignment problem here, as it passes on reinforcement from one agent to another, in accordance with the role of each in obtaining the reinforcement (in terms of their actions or control actions).

Cooperation among members is forged through sharing of reinforcement. An agent calls upon another team member when such an action leads to higher reinforcement for itself (and for other upstream or downstream agents at the same time). This bidding scheme encourages rational and cooperative actions that maximize the total reinforcement from the world (while minimizing the number of steps through discounting).

While members of a team interact and cooperate with each other through bidding, at the same time, they also perform individual reinforcement learning. With this dual process, the whole multi-agent team learns to form "mixed" action sequences carried out by multiple agents alternately in cooperation with each other.

Note that each (Q or CQ) module is implemented with a backpropagation (BP) network. Each BP network consists of three layers of units: input, hidden, and output, as usual. The numbers of input and output units are determined by domain characteristics and encoding requirements. The number of hidden units is set as a parameter, along with other parameters such as learning rate and momentum.

Action selection by an active member agent (the member in control) is conducted based on Boltzmann distributions constructed from Q values. That is,

$$prob(k) = \frac{e^{v_k/\tau}}{\sum_l e^{v_l/\tau}}$$

where τ is the temperature that determines the degree of randomness in action selection. That is, the higher the value of an action is, the more likely the action will be selected. For a Q module, k ranges over all possible actions for the current state s, and the value v_k is simply the Q value for action k in the current state s. For a CQ module, k may be either $continue$ or end, and the value v_k is the CQ value for action k in the current state s.

Evolution may be used to enhance cooperation. We start with an initial population of teams of (random) agents. After a period of training, a genetic algorithm is applied to the population of these teams. A new population (i.e., a new generation) of teams is then formed. The cycle repeats itself.

1. Randomly generate a population of n teams.
2. Train each team in the current population for a certain 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 (weight exchange across two teams). The crossover rate (the percentage of the weights that are exchanged) is β_1. [5]
   (c) Apply mutation to these newly generated teams. The mutation rate (the probability of a weight changing to a random value) is β_2.
4. Replace the current population with the selected m teams and the newly generated $n - m$ teams.
5. Go to step 2.

In tournament selection, the more fit an individual is, the more likely it will be selected. Tournament selection usually goes as follows: Randomly divide all teams into groups of a certain size. In each group, evaluate the fitness of each

[5] γ percent of all crossover is based on the weight exchange at corresponding positions of the two teams. 100-γ percent of all crossover is based on the weight exchange at random pairs of positions.

team by playing them against each other. Select the best performing team in each group to form a new set. Repeat the above steps until m teams (or less) remain (where m is the desired number). In our experiments, however, for the sake of simplicity, we instead played each team against a common benchmark agent and made selections on that basis. [6]
Using the genetic algorithm allows teams to escape local optima more easily. With the algorithm, knowledge not only is exchanged between members in a team, but also is exchanged between teams, so as to allow more exploration.
Note that, although their constitutions are essentially the same, different agents, and different teams, may perform differently because of (1) the intrinsic stochasticity of the model, and (2) the likely stochasticity of the domain the model is applied to. The model is intrinsically stochastic because of the randomness in selecting an action and in selecting an agent within a team (see the specifications above). Some domains may also introduce stochasticity; for example, in Backgammon, the roll of a dice produces random outcomes.

III. Experiments

In this section, we apply MARLBS to a number of domains. These domains are well known and have been tackled before. Therefore, the use of these domains enables us to compare our approach with existing approaches in a quantitative way. There are very few domains used in multi-agent research that are intrinsically multi-agent. Most multi-agent problems may be tackled with one single agent. Viewed from this perspective, the choice of the following domains is justified.

A. Experiments with Learning Backgammon

We applied MARLBS to learning Backgammon, in order to evaluate the model — the usefulness and the performance of MARLBS in complex problem solving (i.e., in complex sequential decision tasks; Qi and Sun 2003). One of the research areas and the application domains of artificial intelligence is the programming of computers to play board games, such as Chess, Checker, GO, and Backgammon. Each of these games has a finite state space with a well-defined set of rules. However, it is usually impossible to search exhaustively a state space. AI research in board game domains has primarily focused on finding satisficing solutions, not optimal solutions.
Backgammon is a board game for two players. See Figure 3 for a depiction of a Backgammon board. Briefly, each player has 15 checkers on a board consisting of 24 spaces. The checkers are moved according to rolls of a dice. Each player tries to bring his/her own checkers home and bears them off before the opponent does, hitting and blocking the enemy checkers along the way. The game has generally been viewed as highly complex, and thus is a good domain for evaluating our learning model.

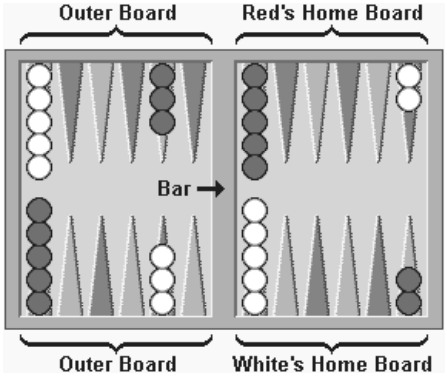

Figure. 3: A Backgammon Board with the checkers in their initial positions.

PubEval is a publicly available computer player (Tesauro 1992). It is a moderately good player, commonly used as a benchmark. We used it as an evaluator of our model. [7]
To evaluate various model components, we also tested them against another benchmark agent — the best single agent chosen from 15 agents after 4000-game training.
Our experimental setup was as follows. The BP networks needed a simple scheme to encode information. TD-Gammon's simple encoding scheme (Tesauro 1992) was adopted. (We tested various encoding schemes but we will describe only one of them, as they did not make any significant difference in terms of results.)
First, let us look into the encoding scheme for input to both Q and CQ modules. For each side of a game, for each position on the board, an encoding with four units was used. The first three units were used to indicate whether there was one checker, two checkers, or three checkers at a position, while the fourth unit indicated that the number of checkers at a position was more than 3. A total of 96 units was used to encode the information at locations 1-24. In addition, for each side, 2 units were used to encode the number of checkers on the bar and off the board. This encoding scheme thus used 98 units for each side. In addition, 12 units were used to encode dice numbers, and additional 16 units to encode one's own first move in a round. Thus, we used a total of 224 input

[6] Although tournament selection costs running time, the time spent can be counted as part of the training time.

[7] Tesauro (1992) developed the best available machine learning program for Backgammon. However, it is not publicly available.

units. [8] [9] Next, the output encoding scheme for Q modules used 16 units. Among these units, 1 to 15 were the checker numbers. A "1" indicated a move by that piece, while "0" meant no action. The output encoding of a CQ network was straightforward, with each option ("end" or "continue") encoded by one unit. The number of hidden units of the BP network for an action module (Q) were 40 and the number of hidden units of the BP network for a controller module (CQ) were 16.

The parameter settings were as follows: the Q value discount rate was 0.95, the learning rate was 0.5, and the temperature was 0.50. The initial weights of BP networks were randomly generated. There were 15 teams in each generation. Each team consisted of 5 agents. Each team of each generation was trained for 200 games. Then a new generation of agents was produced using crossover and mutation. The mutation rate (as defined earlier) was 0.05 and the crossover rate (as defined earlier) was 0.20. [10] The training of teams was done through a mixture of playing against oneself (more than 2/3 of all the games) and playing against PubEval (less than 1/3 of all the games, for the purpose of evaluating MARLBS against PubEval). [11]

A player received payoffs at the end of a game. The payoffs were as shown in Table 1. If the losing side has borne off at least one checker, the rewards for the winner and the loser will be 0.3 and -0.3, respectively. If the loser is "gammoned" (i.e., has not borne off any of the checkers), the rewards for the winner and the loser will be 0.6 and -0.6, respectively. If the loser is "backgammoned" (has not borne off any of the checkers and still has a checker on the bar or in the winner's home board), the rewards for the winner and the loser will be 1.0 and -1.0, respectively.

	Winner	Loser
Backgammoned	+1.0	-1.0
Gammoned	+0.6	-0.6
Other	+0.3	-0.3

Table 1: The payoff table for the backgammon game

We played MARLBS against PubEval. The result of 400,000 iterations was as shown in Figure 4, where an "iteration" is defined as training each of the teams by one game. The maximum, average, and minimum winning percentages were calculated from the 15 teams in each generation. To generate these numbers, each team of each generation played against PubEval for 50 games. The average winning percentage of MARLBS (averaged over all 15 teams) reached 54.8. The maximum winning percentage of MARLBS reached 62.

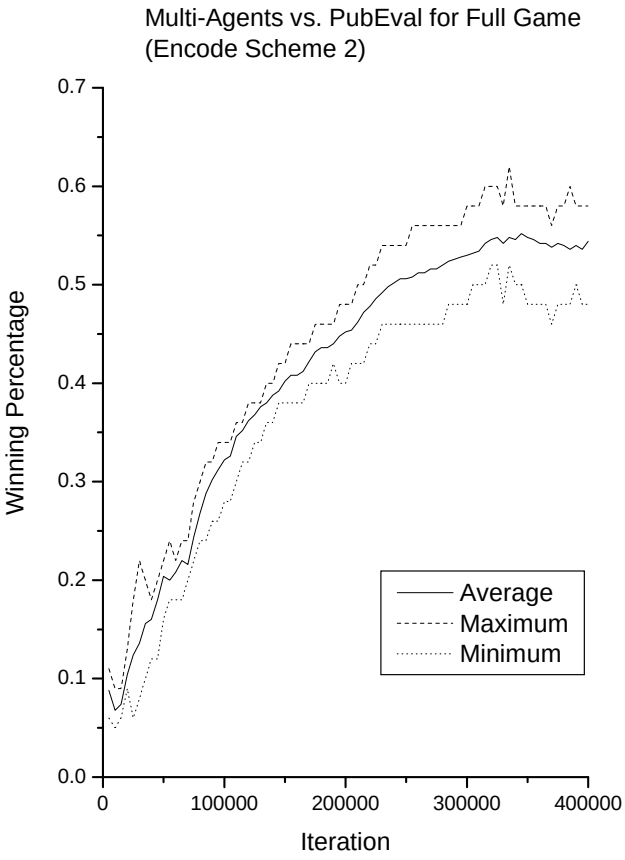

Figure. 4: The winning percentages of MARLBS over time, played against PubEval.

We tested the performance of different teams at different points. The results were as shown in Table 2. In the table, the best/worst teams were those with the highest/lowest performance from a population of 15 teams. The numbers were the winning percentages of a team playing against PubEval for 50 games. The result of the best team playing against the worst team was also included.

As shown in the table, the performance of all the teams improved over the course of training and evolution. Also shown in the table was the fact that the difference between the best team and the worst team decreased over time. At iteration 100,000, the winning percentage of the best team playing against the worst team was 76, while the winning percent-

[8] The reason we needed to encode a player's first move is that, at each round of the game, a player needs to move twice. However, the network only output one move each time. When the player makes its second move in a round, it should consider its first move at the current round. So we encode the player's first move as input into the network.

[9] Other complications in Backgammon, such as doublets, can all be handled within this framework.

[10] 80 percent of crossover was done by the weight exchange at corresponding positions of two corresponding BP networks and 20 percent by the weight exchange at randomly chosen pairs of positions.

[11] Playing against PubEval may lead to exploiting specific weaknesses of PubEval. However, since PubEval was used less than 1/3 of the time, the resulting players should be fairly general. Those episodes were originally conducted for the purpose of generating performance measures of teams against PubEval and we did not want to discard the results of learning during those episodes in order to avoid waste.

Iteration	Best 5 teams			All Teams		
	Avg	Best	Worst	Avg	Best	Worst
100,000	0.32	0.36	0.28	0.30	0.36	0.20
200,000	0.42	0.46	0.36	0.41	0.46	0.28
300,000	0.50	0.54	0.46	0.47	0.54	0.40
400,000	0.51	0.60	0.48	0.49	0.60	0.42

Iteration	Best vs. Worst
100,000	0.76
200,000	0.72
300,000	0.62
400,000	0.58

Table 2: Team Performance in the Backgammon game, measured by winning percentages out of 50 games. Except the data of the best team versus the worst team, all data are for a team playing against PubEval.

age decreased to 58 after iteration 400,000. This reduction of differences was what one would expect, taking into consideration the use of GA here, which selected good teams and discarded bad ones. [12]

The performance of individual team members was also tested. Recall that each member is an autonomous agent and can play the game by itself. The results are in Table 3 and Table 4. Table 3 shows the team member performance, in terms of the best member, the worst member, and the average of all the members, of the best team (that is, the team that had the highest winning percentage when playing against PubEval). The column "Best vs. Worst" records the performance of the best member of the team playing against the worst. For the sake of comparison, the performance of the best team is also listed there. All the numbers are winning percentages of playing against PubEval for 50 games, except the column "Best vs. Worst". Table 4 shows the performance of the members of the worst team. The numbers there are similar.

Iteration	Members in the best team				Team
	Avg	Best	Worst	Best vs. Worst	
100,000	0.32	0.34	0.28	0.66	0.36
200,000	0.42	0.46	0.38	0.54	0.46
300,000	0.49	0.52	0.44	0.52	0.54
400,000	0.50	0.56	0.46	0.52	0.60

Table 3: Member performance in the best team. All data are winning percentages out of 50 games.

Iteration	Members in the worst team				Team
	Avg	Best	Worst	Best vs. Worst	
100,000	0.20	0.24	0.16	0.72	0.20
200,000	0.36	0.38	0.30	0.62	0.28
300,000	0.40	0.42	0.34	0.62	0.40
400,000	0.42	0.46	0.36	0.58	0.42

Table 4: Member performance in the worst team. All data are winning percentages out of 50 games.

[12] Note that the games played against PubEval for the sake of evaluation in the process of applying GA (50 games each) have been taken into account.

We notice that at the end of training, the best team performed better than any member of the best team, including the best member of the best team. The best team also outperformed its members on average. That is, there is a clear advantage in having a multi-agent team, as opposed to choosing the best agent out of them. Generally speaking, when given a set of agents (each trained with some partial experience of games), there are two ways of utilizing these agents: (1) choosing the best agent and using only its decisions, or (2) trying to combine the expertise of all the agents and using their combined decisions (Breiman 1996). Our data show that the latter approach may be better. [13] In some way, this fact shows why multi-agent cooperation may be advantageous, which is due to the synergy within a team, created by the emergent cooperation and coordination of team members. However, notice also that the worst team did not perform better than all its members. This fact suggests that the best teams were able to learn to achieve good coordination among its members, while the worst team failed to do so. It also suggests that the best team was the best at least in part because it achieved better cooperation among its members.

Iteration	Members vs. the best team		
	Avg	Best	Worst
100,000	0.45	0.62	0.38
200,000	0.46	0.52	0.36
300,000	0.45	0.50	0.38
400,000	0.44	0.48	0.40

Table 5: Members of the best team versus the best team. All data are winning percentages of a member playing against its team in 50 games.

Iteration	Members vs. the worst team		
	Avg	Best	Worst
100,000	0.48	0.56	0.32
200,000	0.47	0.58	0.42
300,000	0.46	0.54	0.38
400,000	0.46	0.52	0.36

Table 6: Members of the worst team versus the worst team. All data are winning percentages of a member playing against its team in 50 games.

We also tested the performance of a team playing against its members. The results are in Table 5 and Table 6. For the best team, the performance of the whole team was not better than that of its best member at the beginning. But after a sufficient number of iterations, the whole team outperformed its best member (and all the other members as well). On the other hand, for the worst team, the performance of the whole team was never as good as its best member. Again,

[13] Of course, each agent experienced only a portion of all the games. Thus here we are not comparing a team with an agent having the experience equal to the sum total of all the experiences of the team members. Instead, we are emphasizing the synergy resulting from the cooperation of multiple agents (each with partial experience).

this set of data suggests the following two points: (1) A good team, due to the emergent cooperation of its members, had the advantage of the synergy among agents, and as a result, it performed better than the individual agents on the team; (2) the best team was the best at least in part because of the coordination and cooperation of the team members — these agents learned to cooperate with each other and divided up the task among themselves to achieve a better performance than any individual agent did.

Note that team cooperation improved over time. As indicated by Table 5, early on, the best member of the best team outperformed the team. But after 400,0000 iterations, the team outperformed the agents (including the best agent) on the team. This fact indicated the increasing cooperation of agents on the team over time, which led to the improved team performance against its members (because, without increasing cooperation, a team would never outperform its member agents).

Why does cooperation lead to better performance? Divide-and-conquer is generally a useful approach. In this case, the division of labor among team members makes the learning task faced by each neural network (within each member agent) easier, because each of them can focus on learning a subset of input/output mappings, rather than treating all of them as equally important. In this way, each of them may learn better and, through their cooperation, the whole team may perform better as well. Cooperation in this model emerges from the interaction of agents, and is not externally determined. [14]

In further pursuance of the issue of the synergy of the whole out of its parts, we would like to know, among the three components of our system, GA, RL (i.e., Q-learning), and bidding, which is important, and whether there is any synergy among these algorithms as well. In order to answer this question, four variants (i.e., reduced versions) of MARLBS were tested. First, 15 agents were separately trained for 4000 games each. Then, teams were formed by randomly choosing agents from these (slightly) trained ones. All variants were tested in the same way, by playing against a benchmark agent, which was the best single agent chosen from the 15 agents (after separately training each for 4000 games). [15] Data were then collected from one single run (for the algorithms involving multiple teams, the best team was used).

In variant 1, GA was not applied. Only one team (with RL and bidding) was used. In variant 2, GA and RL were applied, but there was no bidding. That is, each team was reduced to one agent. In variant 3, only GA was applied. Since there was no RL in this variant, each BP network only had the forward phase but not the backward phase (which meant that there was no BP learning although there were BP networks). In variant 4, only RL (with BP networks) was applied. The results for variant 1, 2, 3, 4, and the full MARLBS model are shown in Figure 5(a), 5(b), 6(a), 6(b), and 6(c), respectively. [16]

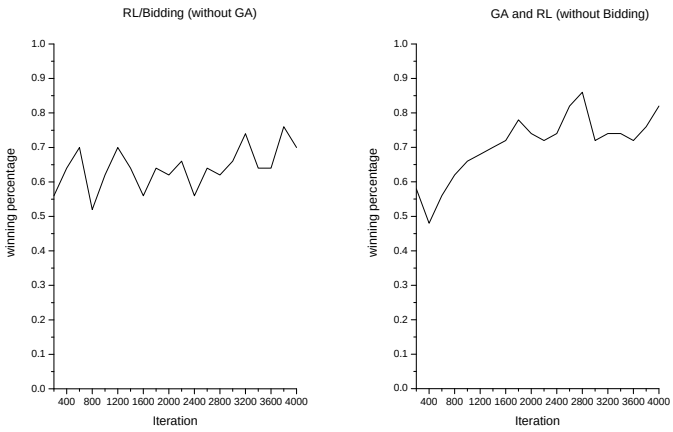

Figure. 5: The winning percentages out of 4,000 games playing against the benchmark agent. (a) RL+bidding (without GA) (b) GA+RL (without Bidding)

From Figure 5 and Figure 6, we can see that variant 4 had the worst average winning percentage. This seemed to indicate that GA and bidding, which forged cooperation among agents of a team, were important in MARLBS. (Separately, we tried running RL for a much longer period of time. There was no significant improvement of performance. We also conducted multiple runs under different initial conditions and there was no significant improvement of performance either.) When compared with MARLBS, all variants' performance was worse than that of MARLBS. All variants took longer to achieve 80% winning (if ever). However, it appeared that all components in our model were useful. Missing any component led to worse performance.

The numerical comparison between MARLBS and its variants is shown in Table 7. Again, we see that variant 4 had the worst average winning percentage. We also see that variant 2 had the best performance among the variants. However, it was worse off than MARLBS. In the table, all variants' average winning percentages and highest winning percentages

[14] An alternative way of describing the advantage that teams have is that different Q functions handle different regions of the state space, which makes the function approximation of each Q function simpler. As a result of appropriate partitioning of the state space, the whole system performs better. See Sun and Peterson (1999) for some analysis.

[15] The reason for using such a benchmark agent was because playing against PubEval would require a lot more training of these variants, which was hard to do given the severe limitation of our computational resources at the time.

[16] The initial performance differences among these variants were in fact not huge. At this point of training, they were all very week, and the differences among them were small (as measured by playing against PubEval). The initial performance differences among these variants were inevitable. For one thing, each game is stochastic (due to dice rolls), and reinforcement learning is stochastic (in its action selection). Therefore, even identical programs perform differently, let alone different programs. The way these variants were formed, as described above, also contributed to the initial differences.

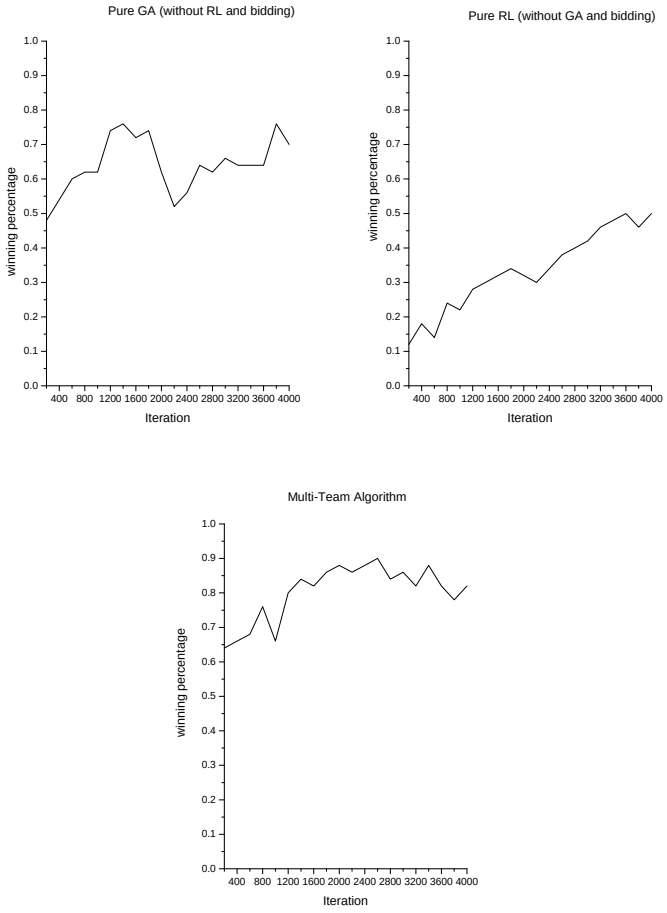

Figure. 6: The winning percentages out of 4,000 games playing against the benchmark agent. (a) GA (without RL and bidding) (b) RL (without GA and bidding) (c) the full system of MARLBS

were lower than that of MARLBS, which indicated that all three components in our system, GA, RL and bidding, were useful. They were synergistic, in the sense that missing any component led to worse performance. The comparison also indicated an interaction between learning and evolution, in the sense that learning enhanced evolution (e.g., by comparing pure GA with GA+RL) and vice verse (by comparing pure RL with GA+RL) (see more discussions of this point later).

Methods	Best	Avg	Avg-1000
RL+bidding (without GA)	0.76	0.64	0.69
GA+RL (without bidding)	0.86	0.71	0.75
GA (without RL/bidding)	0.76	0.64	0.67
RL (without GA/bidding)	0.50	0.34	0.47
MARLBS	0.90	0.80	0.83

Table 7: Comparisons of MARLBS with its variants. The numbers are the winning percentages of playing against a benchmark agent. Avg-1000 indicates the average winning percentages during the last 1,000 games playing against a benchmark agent.

	MARLBS	TD-Gammon	HC-Gammon
Winning %	62	59.3	45
# Iterations	400,000	$> 1,000,000$	400,000

Table 8: Comparisons with other Backgammon players in terms of winning percentage against PubEval.

We may also compare the result of our model with other learning systems. The best known computer player is TD-Gammon (Tesauro 1992). However, it incorporated a number of hand-crafted expert-knowledge features, including concepts such as the existence of a prime, probability of blots being hit, and probability of escaping from behind the opponent's barrier. Pollack and Blair (1998) used feed-forward network representation with hill-climbing search to develop yet another backgammon player (HC-Gammon). [17]

The result of experimental comparisons with these other Backgammon learning systems, in terms of winning percentage against PubEval, is shown in Table 8. MARLBS compares favorably with other Backgammon learning systems. Compared with these other systems, our model achieved a better performance than HC-Gammon, but a roughly comparable performance as TD-Gammon. However, note that MARLBS was trained far less than TD-Gammon (as indicated in Table 8). [18]

Overall, is the extra cost of forming a multi-agent team worth the gain in performance? Note that cost-benefit trade-offs are always tricky. Where the cutoff point on the cost-benefit

[17] Other related work includes Sanner et al (2000) and Darwen (2001).

[18] The process of training and evolution was time-consuming: It took a total of a year and half on the outdated 200 MHz PC we used for the experiments. Thus we had to stop training at the current point to wrap up the project.

curve should be is dependent on the specific goals and specific objectives one has for the system/algorithm in question. There is, generally speaking, no uniform answer to such questions, although we felt it was well worth it.

B. Experiments with TSP

In order to further validate our model beyond the Backgammon domain, we needed to try it on other tasks. One task that stood out was the traveling salesman problem (TSP), which was used by many existing learning algorithms and thus served as a good test domain (Qi and Sun 2005). In our experiment, we chose to use Eilon50 (for 50 cities) and KroA100 (for 100 cities) problems (see, e.g., Eilon et al 1969). The shortest known distance for Eilon50 is 425 and for KroA100 21282. In this domain, through experiments, we reached essentially the same conclusion as we discussed earlier with regard to Backgammon.

We view this task as a sequential decision task, because we may select one city to go to next at each step. Thus, MARLBS (with Q-learning) may be applied. The input and the output of MARLBS was as follows: 3 units were assigned to represent each city. The first unit indicated if the city was the starting city, the second unit indicated if the city was the current city, and the third unit indicated whether the city had been visited before or not. Each output unit represented a city, indicating whether it should be visited next or not. This is the simplest and most straightforward encoding scheme, without any hand-crafted special features to make learning easier.

For Eilon50 (involving 50 cities), a total of 150 input units was thus used to encode the information about the cities. The number of hidden units in the BP networks was 100. The number of output units was 50. For KroA100 (involving 100 cities), a total of 300 input units were used to encode the information about cities. The number of hidden units were 200. The number of output units was 100.

In learning this problem, MARLBS' parameter settings were essentially the same as in the Backgammon game. The Q value discount rate was 0.95, the learning rate was 0.5, and the temperature 0.50. The mutation rate was 0.05 and the crossover rate was 0.20. [19] There were 15 teams in each generation. Each team consisted of 5 agents. A new generation was produced (using crossover and mutation) after every team of the current generation was trained for 100 episodes. The payoff in this case, given at the end of an episode, was set as follows:

$$\text{payoff} = [Optimal/C_i]^2$$

where C_i was the current distance computed at episode i and $Optimal$ was the shortest known distance. [20]

[19] 80 percent of crossover was weight exchange at corresponding positions and 20 percent at random position pairs.

[20] Although this payoff function involved task-specific information (i.e., $Optimal$), our experiments showed that such information was not necessary. An arbitrary value worked just as well.

For Eilon50, all the teams trained with MARLBS reached the shortest known distance (425), taking approximately 11,000 episodes. For KroA100, all the teams trained with MARLBS reached the shortest known distance (21282), taking approximately 200,000 episodes.

We tested the performance of different teams resulting from MARLBS. The results were as shown in Table 9. (The results shown were distances of tours. In order to differentiate the performance of these teams, we only listed their pre-convergence performance, at episode 5,000 and 10,000.) There were a total of 15 teams, with the best and the worst being the ones with the shortest distance and the longest distance, respectively. As shown by the table, while the performance of all teams went up over time (over training episodes, along with evolution), the performance differences among teams were not huge (although there were some differences).

Itn	Best 5 teams			All teams		
	Avg	Best	Worst	Avg	Best	Worst
5,000	642.6	568.4	721.2	674.9	568.4	748.3
10,000	496.5	468.2	568.4	564.3	468.2	621.2

Table 9: Team Performance for the TSP problem. The numbers are the distances of tours achieved.

To see the advantage of using teams of agents rather than single agents, we compared the performance of a team with that of its member agents. The results were as shown in Table 10 and Table 11. Performance was measured by tour distances. Table 10 shows the performance of the members of the best team. Table 11 shows the performance of the members of the worst team.

Iteration	Members in the best team			Best team
	Avg	Best	Worst	
5,000	679.1	568.4	748.3	568.4
10,000	537.8	520.2	598.4	468.2

Table 10: The performance of the members of the best team and the best team as a whole. The numbers shown are the distances achieved.

Iteration	Members in the worst team			Worst team
	Avg	Best	Worst	
5,000	898.0	721.2	1,011.2	721.2
10,000	644.9	520.2	721.2	621.2

Table 11: The performance of the members of the worst team and the worst team as a whole. The numbers shown are the distances achieved.

As clear from the tables, at iteration 10,000, the performance of the best team was better than that of its best member (and all the other members). As in the Backgammon experiments, this fact again demonstrated, in the context of the TSP problem, the advantage of cooperative teams with division of labor among team members. The performance of the worst team was no better than its members, which suggested that

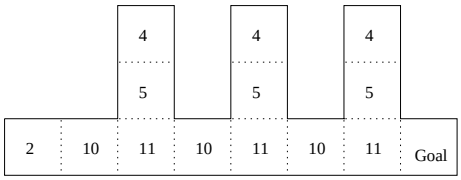

Figure. 7: A maze requiring segmentation and reuse of agents. Each number indicates a unique observational state as perceived by agents.

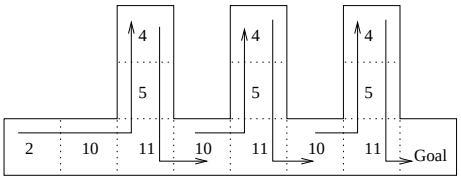

Figure. 8: A segmentation using a team of only two agents. The two agents of the team alternate.

there was no good cooperation among its member agents. It appears that cooperation and synergy among members were the key for a good performance by a team.

C. Experiments in Dealing with POMDP

Sun and Sessions (1999) proposed the segmentation of action sequences through the cooperation of a team of agents in sequential decision tasks. A multi-agent team was formed through bidding, without involving GAs. It segmented action sequences, for the purpose of dealing with partially observable Markov decision processes (POMDP). Segmentation was based on reinforcement received during task execution, with different agents communicating with each other through sharing reinforcement estimates obtained by each other (i.e., through bidding), as in other applications of MARLBS. It did not rely on a priori knowledge or a priori structure in dealing with POMDP.

Let us examine an example. In one maze (Figure 7), there is one starting location ("2") at one end and one goal location at the other. However, before reaching the goal location, the agents have to reach the top of each of the three arms. The agents receive local information concerning the four adjacent cells at the current location regarding whether each is an opening or a wall. Thus, each observational state is made up of 4 bits. This is the simplest encoding one can find. See Figure 7, where each number indicates an unique observational state as perceived by agents. As indicated by the numbers, several cells may be perceived to be the same because they have the same adjacent cell (walls or openings) configuration.

The team can make a move to any adjacent cell at each step (either to go left, go right, go up, or go down). But if the adjacent cell is a wall, it will remain in the original cell at the end of the step. The payoff for reaching the goal location (after visiting the tops of all the three arms) is 1.

In this domain, the shortest path consists of 19 steps. A minimum of two agents are needed on a team, in order to remove non-Markovianness through segmentation and thereby to obtain the shortest path. A single agent with atemporal representation could not learn the task, because there was no consistent, deterministic Markovian policy possible. With two agents there is exactly one way of segmenting the sequence (when only considering the shortest path to the goal): switching to a different agent at the top cell of each arm (marked as "4") and switching again at the middle cell between any two arms (marked as "10"). See Figure 8. This segmentation involves repeated use of the same agents along the way to the goal. [21]

The parameter settings were as follows: The Q value discount rate was 0.97, the initial learning rate was 0.9, the learning rate changed according to $\alpha^t = \alpha^0/t^{1/5}$ (where t was the number of episodes completed), the initial temperatures were $\tau_Q^0 = 0.6$ (for the Q modules) and $\tau_{CQ}^0 = 0.8$ (for the CQ modules), and the temperatures changed according to $\tau^t = \tau^0/t^{1/2}$. No GA was used. The number of steps allowed in each episode during learning was 200 (an episode ended when the limit was reached, or as soon as the goal was reached after having reached the tops of all the three arms). Training lasted for 10,000 episodes in each case, with a different number of agents on the team in each case.

In this domain, learning did not become easier when we added more agents to the team. This was because there was only exactly one way of segmentation that might lead to an optimal path, and thus there was no advantage in using more agents. An ANOVA analysis (number of agents x block of training) confirmed this conclusion. Using two agents, we achieved a 62% optimal path rate, after 10,000 episodes of training. See Figure 9 for an example of the learned agents of a team (their Q and CQ modules). See Figure 10 for the performance data.

Experiments in this domain demonstrated the points regarding the feasibility of segmentation of action sequences through a team dividing up a sequence. Furthermore, experiments in this domain demonstrated the reuse of agents (in several different places) in a sequence or, in other words, the possibility of agents acting like subroutines that could be called into use any number of times. Reuse of agents not only led to the "compression" of action sequences, it also allowed the handling of action sequences that could not be (efficiently) handled otherwise (cf. Wiering and Schmidhuber 1998). A POMDP was divided up into two MDPs (Markov Decision Processes) (Sun and Sessions 2000). This is another form of team cooperation, further extending the cases we discussed earlier.

[21] However, when more agents are added, reuse might be reduced: a third agent, for example, can be used in place of the second use of agent 1 (see Figure 8).

```
State 2
 Agent #0  Q:   UP(0.00)  **RIGHT(0.60)    DOWN(0.00)    LEFT(0.00)
          CQ:           **CONTINUE(0.70)     END(0.03)
 Agent #1  Q:   UP(0.00)  **RIGHT(0.30)    DOWN(0.00)    LEFT(0.00)
          CQ:           **CONTINUE(0.04)     END(0.03)
          AQ: **0(0.72)   1(0.02)
State 4
 Agent #0  Q:   UP(0.00)    RIGHT(0.00)  **DOWN(0.61)    LEFT(0.00)
          CQ:           CONTINUE(0.70)   **END(0.74)
 Agent #1  Q:   UP(0.00)    RIGHT(0.00)  **DOWN(0.66)    LEFT(0.00)
          CQ:           **CONTINUE(0.74)     END(0.01)
          AQ:   0(0.70) **1(0.74)
State 5
 Agent #0  Q: **UP(0.62)    RIGHT(0.00)    DOWN(0.61)    LEFT(0.00)
          CQ:           **CONTINUE(0.70)     END(0.12)
 Agent #1  Q:   UP(0.64)    RIGHT(0.00)  **DOWN(0.66)    LEFT(0.00)
          CQ:           **CONTINUE(0.67)     END(0.10)
          AQ:   0(0.04) **1(0.05)
State 10
 Agent #0  Q:   UP(0.00)  **RIGHT(0.75)    DOWN(0.00)    LEFT(0.60)
          CQ:           **CONTINUE(0.80)     END(0.04)
 Agent #1  Q:   UP(0.00)    RIGHT(0.40)    DOWN(0.00)  **LEFT(0.66)
          CQ:           CONTINUE(0.74)   **END(0.79)
          AQ: **0(0.80)   1(0.03)
State 11
 Agent #0  Q: **UP(0.62)    RIGHT(0.29)    DOWN(0.00)    LEFT(0.29)
          CQ:           **CONTINUE(0.72)     END(0.11)
 Agent #1  Q:   UP(0.64)  **RIGHT(0.68)    DOWN(0.00)    LEFT(0.29)
          CQ:           **CONTINUE(0.77)     END(0.09)
          AQ: **0(0.15)   1(0.10)
```

Figure. 9: The Q and CQ modules of the resulting agents on the team after learning. "**" indicates the best action in a state.

# of Agents	% of Opt. Path $\tau = 0$	% of Opt. Path $\tau = 0.001$	Avg Path Length $\tau = 0.001$	Pairwise t Test $T(> 2.33)$
2	60%	52%	22.51	
3	43%	38%	23.85	8.600
4	27%	20%	24.85	4.010
5	27%	22%	25.42	1.281

Figure. 10: The effect of number of agents on the performance (after learning).

Note that other POMDP cases have been handled by MARLBS before as well. See Sun and Sessions (1999, 2000) for more details. As discussed earlier, the present work extends the method beyond dealing with POMDP, and also adds evolution to further enhance cooperation among member agents of a team.

IV. Discussions and Comparisons

Below, we will discuss a few issues that are highly relevant to our model. In the process, we will review some relevant prior work, and their relations to the work reported here.
Learning and evolution have been studied extensively in the research communities of machine learning, neural networks, and evolutionary computation. However, co-learning and co-evolution, which are interesting research issues in multi-agent systems, have been studied much less. More attention, in the form of either theoretical or experimental studies (as in this work), is needed. Furthermore, the interaction of learning and evolution is also interesting, given the fact that we barely begin to explore such a phenomenon. Below, let us look into some details.

A. Co-Learning from Reinforcement

Although co-learning using reinforcement learning algorithms have had a fairly long history, the issue has proved to be difficult.
Andy Barto and associates explored cooperativity in reinforcement learning very early on (see, e.g., Barto et al 1983). Another piece of early work is Tan (1993), who conducted multi-agent Q-learning experiments. In his task, there were several prey and predator agents. Predators had limited vision so that they might not always know where preys were. Thus the predators could help each other by informing each other of their sensory input. They could also help each other by exchanging reinforcement episodes and/or control policies. This was an example of using built-in cooperation strategies in co-learning situations. In general, we may not be able to assume any cooperative inclinations on the part of agents. The general co-learning problem is more difficult.
Sandholm and Crites (1995) used Q-learning agents to play the IPD game. In their experiments, Q-learning agents learned the optimal strategy when playing against a fixed strategy player. However, Q-learning agents had difficulty when playing against other Q-learning agents. Sen and associates explored cooperation using reinforcement learning and achieved some limited success (see e.g. Sen and Sekaran 1998). Sun and Qi (2000) explored the issue of rationality assumptions on the part of agents and their effects on co-learning. Littman (2001) presented some theoretical results on multi-agent reinforcement learning in zero-sum games. Bowling and Veloso (2002) also presented some theoretical results on modified Q-learning algorithms. However, in general, co-learning using Q-learning is difficult and lacks theoretical guarantees, which makes practical approaches such as

the one developed in this paper useful.
Note that, relatedly, there are multi-agent reinforcement learning algorithms in the literature that extend simple reinforcement learning (such as Q-learning). For example, Sun and Peterson (1999) divided up complex reinforcement learning tasks to accelerate learning. A state space was automatically partitioned into multiple regions. Different agents were assigned to these regions respectively. Some heuristics were proposed to decide on how to partition spaces and assign agents to subspaces. This partitioning approach reduced the learning complexity of each agent, and thus the overall learning difficulty. Sun and Sessions (2000), as mentioned earlier, extended this approach by partitioning not state spaces but action sequences. These approaches are limited in terms of their scopes and performance. [22]

B. Bidding

Algorithms for bidding constitute a key issue in auction theory. Generally speaking, an auction is a market mechanism with (explicit or implicit) rules for allocating resources and determining prices on the basis of bids from market participants (see Hendriks and Paarsh 1995, McMillan and McAfee 1987). In typical auctions, there are one seller and a group of competing buyers who bid to possess the auction objects. Procurements, on the other hand, refer to situations in which a buyer wishes to purchase objects from a set of potential suppliers. There are various forms of auctions (for example, the English auction, the Dutch auction, the first-price sealed-bid auction, the second-price sealed-bid auction, and so on).
Based on the idea of bidding, Baum and Durdanovic (2000) developed a system of artificial economy (named *Hayek*) to learn to solve the block world problem (as well as other problems). It consisted of a collection of agents, each consisting of a program with an associated numeric wealth. The system embodied two economics principles: property rights and conservation of money. Useful agents received payoffs (from the external world or from other agents) and thus accumulated wealth, while less useful ones lost their wealth and disappeared. In this way, the system evolved into a solution for the problem.
Bidding algorithms may also be an efficient method for allocating resource, applicable to distributed computing and networking. For example, Gagliano et. al. (1995) applied bidding to allocate decentralized network resources. There are many other similar proposals related to information technology.
In MARLBS, bidding is used to distribute expertise (capabilities of individual agents) within a team. Bidding is made simpler in MARLBS, because of the restriction that each agent can only bid its best Q value. Thus there is no intentional under-bidding or over-bidding. Basically, a simple form of a first-price auction is used here, but with a twist: Stochasticity in bid selection allows "exploration" (trial-and-error learning) within a team. Furthermore, each agent is capable of learning and thus, different from Baum and Durdanovic's model, agents do no simply live or die, but adapt to niches individually. There are also advantages in mixing learning and evolution as in MARLBS, in the form of enhanced performance (as demonstrated in the Backgammon domain by comparing GA+RL with pure RL and with pure GA; see also the discussion later on learning/evolution interaction).

C. Evolutionary Approaches

There has been some interesting work on evolution of cooperative teams. Salustowicz et al (1998) applied co-evolution to a simulated soccer game. In their model, the agents of each team shared the same action set and policy, but might behave differently because of position-dependent inputs. All agents of a team were rewarded or punished collectively. They compared several algorithms, and found that the evolutionary approach achieved the best performance. Nolfi and Floreano (1998) investigated the role of co-evolution in the context of predator and prey games. Such settings may be viewed as a form of "cooperation" in an extended sense.
These results were generally in line with our findings that evolution enhanced team cooperation and thus the overall performance of a team. Although each of the aforementioned models adopted a different evolutionary algorithm, they achieved the same desired effect — enhancing cooperation among team members.
In terms of mixing evolutionary computation and neural networks, there have been a variety of models in the literature. For example, Gomez and Miikkulainen (1999) implemented a neuro-evolutionary system to evolve a controller for the standard double pole task (as well as other non-Markovian control tasks). They showed that their neuro-evolution was faster than other methods. Fogel (2000) used the evolutionary computation and neural networks for learning to play checkers.
Compared with MARLBS, these methods did not use multi-agent teams. It is possible that their performance could be enhanced by adopting a multi-agent approach on top of evolution. We demonstrated experimentally earlier that team cooperation with evolutionary computation is better than the latter alone.

D. Learning/Evolution Interaction

Let us look into an interesting issue — the interaction between learning and evolution in the formation of cooperative teams. Ackley and Littman (1992) was one of the earlier computational studies of this issues. They developed an algorithm with combined evolution and learning, and applied

[22] Note that there have also been a host of hierarchical reinforcement learning models that are somewhat similar to these multi-agent reinforcement learning algorithms. But in these models, external interventions (e.g., in terms of dividing up a state space, or assigning agents to subtasks) are necessary. Thus, these models are not adequately addressing co-learning issues.

it to an artificial environment populated with adaptive and non-adaptive agents. They found that learning and evolution together were more successful than either alone.
Nolfi and Floreano (1999) argued that adaptation in dynamic environments gained a significant advantage by the combination of learning and evolution. The interaction between learning and evolution altered both evolution and learning processes. In MARLBS, this advantage was confirmed — We showed that the combination of learning and evolution led to better overall performance.
Note that in MARLBS, the effect of learning was propagated to offsprings through Lamarckian evolution (that is, through evolution that directly used the result of prior learning). An alternative is to avoid using such a form of evolution by avoiding using the result of learning in evolution. Instead, we may only use the learning ability and the starting condition in evolution. With this alternative, we can still explore the interaction between learning and evolution, albeit through the Baldwin effect in an indirect way (Arita and Suzuki 2000). However, the overall training and evolution process would be much longer in this way.

E. Other Issues

There are other ways of establishing and/or enhancing cooperation among agents. Let us discuss a few of these ways, and their respective advantages and shortcomings, in relation to MARLBS.
Built-in cooperation. There are many multi-agent models with build-in (hard-wired) cooperation. In that case, agents are not completely autonomous — They are dictated to cooperate with each other in some pre-destined ways (as opposed to emergent cooperation out of self interest).
For example, in Korf (1992), a policy was introduced for each predator agent based on an attractive force to the prey and a repulsive force from other predators. Thus the predators tended to approach the prey from different sides. The cooperation was thus built into the agents. Also, as mentioned earlier, in Tan (1993), agents were forced to communicate with each other and thus his model was also a case of built-in cooperation.
Obviously, this approach may establish cooperation among agents more easily. But this approach is not generally applicable. It only works in certain special circumstances in which pre-destined cooperation is feasible. In MARLBS, we tried to minimize such built-in requirements, and thus agents in MARLBS are more autonomous, with a minimum level of pre-determination for cooperation. MARLBS can certainly be applied to these above task domains, where cooperation may emerge from the interaction of agents based on their self interest.
Models of other agents. In Nadella and Sen's (1997) experiments conducted in a soccer simulator, an agent learned about teammate and opponent capabilities through repeated trials of specific actions. Each player had an assigned efficiency (e.g., in the range [0,1]) for the execution of an action (such as passing, tackling, and dribbling) corresponding to the probability that the action would succeed. Agents did not start with knowledge of the abilities of themselves, their teammates, or their opponents. Instead, they learned to estimate these based on repeated trials. The agents could then base their actions on learned parameters.
Hu and Wellman (2001) considered various levels of recursive models for dynamic multi-agent systems. Their experiments showed that the learning agents on average outperformed the non-learning agents that did not use information about others. Wellman and Hu (1998) proposed the notion of a conjectural equilibrium, which was concerned with the situations in which all agents' expectations were realized and each agent responded optimally to its expectations. Agents execute learning strategies with the building of models of responses of other agents.
Sun and Qi (2000) also looked into building models of other agents. They conducted experimental comparisons between the agents who did build models and did make the assumption that other agents were rational and the agents who did not. The comparisons showed that models of (and rationality assumptions about) other agents were often helpful in reaching a more rational outcome (see also Claus and Boutilier 1998).
This approach is useful in terms of establishing cooperation among multiple agents. It, however, incurs a higher cost, due to additional learning about other agents. MARLBS avoided this approach: With Q values as bids, there is no need for learning models of other agents, because Q values represent an accurate picture of what other agents are currently capable of.

V. Concluding Remarks

This paper presents an approach toward forming cooperative multi-agent teams, from the interaction of self-interested agents, based on bidding utilizing reinforcement values acquired through reinforcement learning. The result is a straightforward method that is generic and works in a variety of task domains.
The results and discussions so far illustrated our main point: That is, a cooperative team is advantageous compared with the single agents (including the best of the single agents) making up the team. Our approach toward building cooperative teams has been validated through learning Backgammon, solving TSP, and dealing with POMDP.
In our approach, we make sure that the autonomy and the self interest of individual agents are maintained as much as possible, so that flexibility and generality are ensured, while cooperation among self-interested agents of a team is facilitated.
The key practical question concerning forming a cooperative team of agents is how such cooperation may be achieved among self-interested agents. In this work, we presented our own approach and empirically demonstrated it. Our experi-

mental results thus far appear to indicate the validity and the advantage of the approach.

ACKNOWLEDGMENT: This work was supported in part by ARI grant DASW01-00-K-0012, and a University of Missouri startup fund. We thank Harold Stone for his suggestions early on during this project.

References

[1] D. Ackely and Michael Littman, (1992). Interactions between learning and evolution. In K. R. Brown, editor, *Artificial Life II (Proceedings Volume X in the Santa Fe Institute Studies in the Sciences of Complexity)*, pages 487–507. Addison-Wesley, 1992.

[2] Takaya Arita and Reiji Suzuki, (2000). Interactions between learning and evolution: The outstanding strategy generated by the Baldwin effect. In Eilis Boudreau and Carlo Maley, editors, *Proceedings of Artificial Life VII*, pages 196–205, Portland, OR, 2000.

[3] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, (1983). Neuron-like elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[4] Eric B. Baum and Igor Durdanovic (2000). Evolution of cooperative problem solving in an artificial economy. *Neural Computation*, 12:2743–2775, 2000.

[5] Michael Bowling and Manuela Veloso, (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136, 215–250.

[6] L. Breiman, (1996). Stacked regressions. *Machine Learning*, 24, 49-64.

[7] Cristiano Castelfranchi, (2001). The theory of social functions: challenges for computational social science and multi-agent learning. *Cognitive Systems Research*, special issue on multi-disciplinary studies of multi-agent learning (ed. Ron Sun). Vol.2, No.1, pp.5-38.

[8] C. Claus and C. Boutilier, (1998). The dynamics of reinforcement learning in cooperative multiagent systems. in: *Proc. of International Conference on Autonomous Agents (Agents'98)*.

[9] P. Darwen, (2001). Why co-evolution beats temporal difference learning at Backgammon for a linear architecture, but not a non-linear architecture, *Proceedings of the 2001 Congress on Evolutionary Computation*, 1003-1010.

[10] S. Eilon, T. Waston-Gandy, and N. Christofides, (1969). Distribution management: mathematical modeling and practical analysis. *Operational Research Quarterly*, 20, 37-53.

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

[12] R. A. Gagliano, M. D. Fraser, and M. E. Schaefer, (1995). Auction allocation of computing resources. *Communications of ACM*, 38:88–99, 1995.

[13] Faustino Gomez and Risto Miikkulainen, (1999). Solving non-markovian control tasks with neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99)*, Stockholm, Sweden, 1999. Morgan Kaufmann.

[14] Kenneth Hendricks and Harry J. Paarsh, (1995). A survey of recent empirical work concerning auctions. *Canadian Journal of Economics*, 28:403–426, 1995.

[15] Junling Hu and Michael P. Wellman, (2001). Learning about other agents in a dynamic multiagent system. *Journal of Cognitive System Research*, 2(1):67–79, 2001.

[16] J. Kahan and A. Rapoport, (1984). *Theories of Coalition Formation*. Lawrence Erlbaum Associates, London.

[17] Richard E. Korf, (1992). A simple solution to pursuit games. In *Working papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 183–194, 1992.

[18] Michael L. Littman, (2001). Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, Volume 2, Issue 1, April 2001

[19] J. McMillan and R. P. McAfee, (1987). Auctions and bidding. *Journal Economic Literature*, 25:699–738.

[20] Rajani Nadella and Sandip Sen, (1997). Correlating internal parameters and external performance: learning soccer agents. In Gerhard Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning*, pages 137–150. Springer Verlag, 1997.

[21] Stefano Nolfi and Dario Floreano, (1998). Co-evolving predator and prey robots: Do "arm races" arise in artificial evolution? *Artificial Life*, 4(4):311–335, 1998.

[22] S. Nolfi and D. Floreano, (1999). Learning and evolution. *Autonomous Robots*, Volume 7, Number 1, Pages 89-113.

[23] J. Pollack and A. Blair, (1998). Co-evolution in the successful learning of Backgammon strategy. *Machine Learning*, 32(3), 225-240.

[24] D. Qi and R. Sun, (2003). A multi-agent system integrating reinforcement learning, bidding and genetic algorithms. *Web Intelligence and Agent Systems*, Vol.1, No.(3-4), pp.187-202,

[25] D. Qi and R. Sun, (2005). Learning to cooperate in solving the traveling salesman problem. *International Journal of Neural Systems*, Vol.15, No.1&2, pp.151-162. 2005.

[26] J. Rosenschein and G. Zlotkin, (1994). *Rules of Encounters*. MIT Press, Cambridge, MA.

[27] R. Salustowicz, M. Wiering, and J. Schmidhuber, (1998). Learning team strategies: Soccer case studies. *Machine Learning*, 33:263–283, 1998.

[28] Tuomas W. Sandholm and Robert H. Crites, (1995). Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, special issue on the Prisoner's Dilemma, 37:147–166, 1995.

[29] S. Sanner, J. Anderson, C. Lebiere, and M. Lovett, (2000). 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.

[30] Sandip Sen and M. Sekaran, (1998). Individual learning of coordination knowledge. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Learning in Distributed Artificial Intelligence Systems*, 10:333–356, 1998.

[31] Y. Shoham and M. Tennenholtz, (1994). Co-learning and the evolution of social activity. Technical Report STAN-CS-TR-94-1511, Stanford University.

[32] R. Sun, (2001). Cognitive science meets multi-agent systems: a prolegomenon. *Philosophical Psychology*, Vol.14, No.1, pp.5-28. 2001.

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

[34] R. Sun and D. Qi, (2000). Rationality assumptions and optimality of co-learning. *Design and Applications of Intelligent Agents*. pp.61-75. C. Zhang and V. Soo (eds.), Springer-Verlag, Heidelberg, Germany.

[35] Ron Sun and Chad Sessions, (1999). Bidding in reinforcement learning: A paradigm for multi-agent systems. In *Proceedings of The Third International Conference on Autonomous Agents (AGENTS'99)*, Seattle, WA, 1999.

[36] R. Sun and C. Sessions, (2000). Self-segmentation of sequences: Automatic formation of hierarchies of sequential behaviors. *IEEE Transactions on Systems, Man, and Cybernetics: Part B, Cybernetics*, Vol.30, No.3, pp.403-418. 2000.

[37] Ming Tan, (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on machine Learning*, pages 330–337, 1993.

[38] Gerald Tesauro, (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[39] C. Watkins, (1989). *Learning with Delayed Rewards*. Ph.D Thesis, Cambridge University, Cambridge, UK.

[40] Michael P. Wellman and Junling Hu, (1998). Conjectural equilibrium in multiagent learning. *Machine Learning*, 33:179–200, 1998.

[41] M. Wiering and J. Schmidhuber, (1998). HQ-learning. *Adaptive Behavior*, 6 (2), 219-246.

[42] R. Wilson, (1992). Strategic analysis of auctions. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 1, pages 227–279. Elsevier Science, New York, 1992.

Author Biographies

Ron Sun is Professor of Cognitive Science at Rensselaer Polytechnic Institute, and formerly the James C. Dowell Professor of Engineering and Computer Science at University of Missouri-Columbia. He received his Ph.D in 1992 from Brandeis University. His research interest centers around studies of cognition, especially in the areas of cognitive architectures, human and machine reasoning and learning, multi-agent interaction and social simulation, and hybrid connectionist models. He is the founding co-editor-in-chief of the journal Cognitive Systems Research (published by Elsevier). He also serves on the editorial board of Connection Science, Applied Intelligence, and other journals. He is the general chair and program chair for COGSCI 2006. He is a member of the Governing Board of International Neural Networks Society.

Dehu Qi received his Ph.D in Computer Science from University of Missouri-Columbia in 2002. He is now Assistant Professor of Computer Science at Lamar University in Beaumont, Texas. His research interests include multi-agent systems, genetic algorithms, and machine learning.