Cooperative Agents Through Bidding

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

Ron Sun
Rensselaer Polytechnic Institute
Cognitive Science Department
110 Eighth Street, Carnegie 302A
Troy, New York 12180
rsun@rpi.edu

Abstract

One of the main research topics in Multi-Agent Systems is learning cooperation among agents. This paper presents a multi-agent reinforcement learning approach with bidding. Self-interested agents cooperate with each other through bidding and evolutionary computation. We tested the approach to the TSP problem. The experimental results show our approach can achieve a certain level of performance in problem solving.

1 Introduction

In the multi-agent systems, an individual agent has the power to act and usually has its own interest. The environment usually contains other agents. When building a multi-agent system, it is a key issue to develop agents that cooperate with each other to accomplish a complex task. However, multi-agent cooperation and co-learning is a difficult task. We are interested in how agents learn to cooperate with each others, and the role of cooperation and negotiation in a multi-agent system.

In this paper, we will look into a bidding approach for multi-agent reinforcement learning that learns complex tasks [11]. Section 2 details our approach for multi-agent systems. Section 3 presents and discusses experimental results and section 4 concludes.

2 Our Approach

In our approach, a 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 team will have more chances to be chosen as the member-in-control.

The member agent learns to deal with its environment through 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 receives the reward from the next member-in-control. This is also a form of communication among members.

The type of our system is a homogeneous communicating system since all members have the same structure and have the same ability to receive the environmental information. Furthermore, the method of distributed control in our system is competitive since we use the bidding algorithm in our system.

2.1 Details of the Reinforcement Learning

Each member in each team is a reinforcement learning agent, which is implemented by the modified version of Q-learning algorithm [12]. In our approach, each member(agent) in the team has two modules: the Q module and the CQ module. In our experiments, both modules are implemented by the back-propagation neural networks. Each member can select an action 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

[1]Appeared in the proceedings of the 2004 IEEE International Conference on Information Reuse and Integration (IRI-2004), p444-449, Las Vegas, NV, 2004

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.

In any given state, a member (i.e., a pair of Q and CQ) is in control. When the CQ module in the member selects "continue", the corresponding Q module will select an action with regard to the current state that will affect the environment and thus generate rewards from the environment and incur costs in the environment. When the CQ selects "end", the control is relinquished by the member. A bidding process ensures which proceeds immediately to select another member (a pair of CQ and Q) to take over. This cycle then repeats itself.

Each member decides on its best 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 mutual 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.

The learning rules can be specified 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. When the next action by CQ_k is *end*, the Q_k module receives as reward 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 member will accept (from subcontractors), if it gives up control at this point, the value of which is the expected (discounted) total of reinforcement that will be received from that point on by the whole system. This value is given to the Q module so that it can take it into account when deciding on its courses of action (e.g., whether to reach one giving-up point or another).

- 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 CQ. That is , when CQ_k decides to continue, it accumulates reinforcement generated by the actions of the corresponding Q_k. When the current action by 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 is the Q module of the next member in control (the chosen bidder l) and $\max_a Q_l(s,a)$ is the chosen bid. That is, when a CQ ends, it takes the chosen bid as its reward, which gives it incentive to take higher bids. It learns the expected value of bids, which is the expected (discounted) total reinforcement that will be accumulated by the whole system from the current state on. So in effect, CQ makes its continue/end decisions based on comparing whether giving up control or continuing control will lead to more reinforcement. Thus members are rational in this regard.

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.

2.2 The Multi-Team system

In our experiments, we implemented two algorithms: the multi-team algorithm and the single-team algorithm.

The Multi-Team Algorithm

In the multi-team system, we randomly generate a set of teams and train them for a number of training episodes. In the training process, the team learns by playing against itself. The performance of the Q and CQ module, 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, which will be discussed shortly. 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 two random positions.
   (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.

Tournament Selection

We use the tournament selection algorithm, as following, to select the best teams:

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* remain, where m is the number of members we need.

In our experiments, the fitness value is the winning percentage when a member plays against the benchmark agent. In the 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.

The Single-Team Algorithm

The single-team algorithm is a variant to the multi-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. 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. The detail of the single-team algorithm is as follows:

1. Randomly generate a team.
2. Train the team for a number of episodes.
3. Evaluate the team by comparing it with the old team. If the team's performance is worse than that of the old team, restore the old team.
4. Randomly exchange weights between members. The crossover rate (the percentage of the weights that have been exchanged between two members) is α.
   (a) γ percent of crossover is based on the weight exchange at the corresponding position.
   (b) *100-*γ percent of crossover is based on the weight exchange at two random positions.
5. Randomly mutate selected members. The mutation rate (the percentage of the weights that have been mutated in a member) is β.
6. Replace the team with the newly generated team.
7. Go to step 3.

3 Experimental Results

3.1 Experiment Setup

The TSP problem is a well-known NP problem. Given a set of n cities, the TSP problem can be stated as the problem of finding a minimal length closed tour that visits each city once and only once. Suppose d_{ij} is the distance of the path between city i and j. d_{ij} will be equal to Euclidean distance between city i and j In our experiments.

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

We choose Eilon50 [5] and KroA100 [9] problems for our experiments. The shortest known distance for Eilon50 is 425 and for KroA100 is 21282 [9].

The BP network is used to implement the reinforcement learning algorithm. The encoding method is as follows: 3 units are assigned to represent each city. The first unit is used to encode if the city is the start city, the second unit is used to encode if the city is the current city, and the third unit is used to encode if the city has been visited. For the Eilon50 problem, a total of 150 units is used to encode the cities' information. The number of hidden units is 100. For the KroA100 problem, a total of 300 units is used to encode the cities' information. The number of hidden units is 200.

The initial parameter settings are as follows: the Q value discount rate is 0.95, the learning rate is 0.5 and the temperature is 0.50. The mutation rate is 0.05 and the crossover rate is 0.20. 80 percentage of crossover is the weights exchange at the corresponding position and 20 percentage of crossover is the weights exchange at the random position.

The player will receive reward after it generates a tour. The reward is set as follows:

$$reward = [Optimal/C_i]^2$$

where C_i is the current distance computed at iteration i and Optimal is the shortest distance we known.

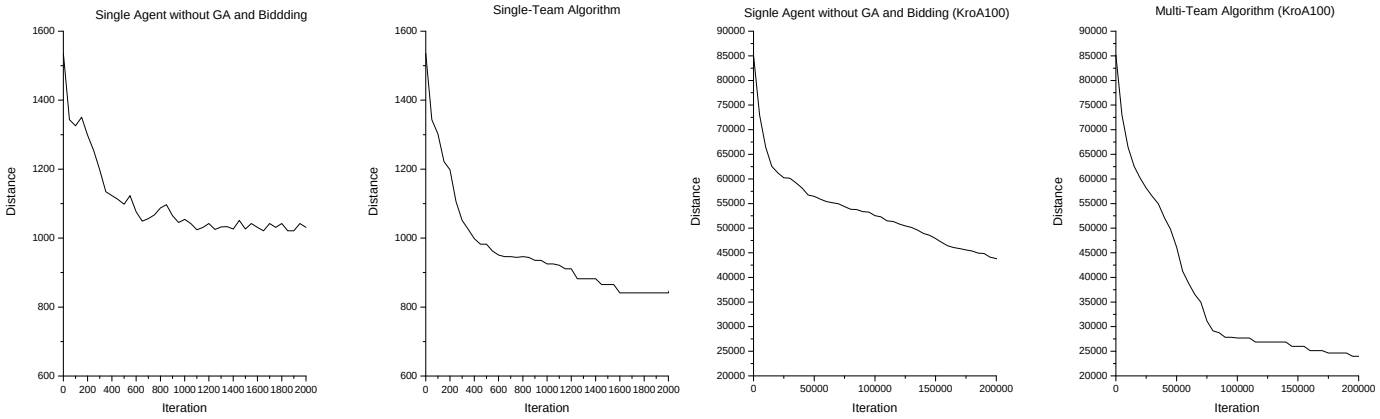
3.2 Experiment Results

For the Eilon50 problem, we tested 3 different algorithms: the single agent, the single-team algorithm, and the multi-agent algorithm. The single agent uses the Q-learning algorithm without GAs and biding.

In the single-team algorithm, the team is formed by selecting the best 5 single agents after 1000 episodes. In the multi-team algorithm, 15 teams are randomly generated at the beginning. After 200 iteration training, 5 teams are selected for next generation. 10 new teams are generated by applying mutation and crossover to these chosen 5 teams. 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. The experiment results of 2,000 iterations for Eilon50 problem are in the Figure 1.

For the KroA100 problem, we tested 2 different algorithms: the single agent and the multi-team algorithm. The experiment results for 200,000 iterations of KroA100 problem are in Figure 2.

The experiment shows again that our multi-agent system approach outperforms the single agent. For Eilon50



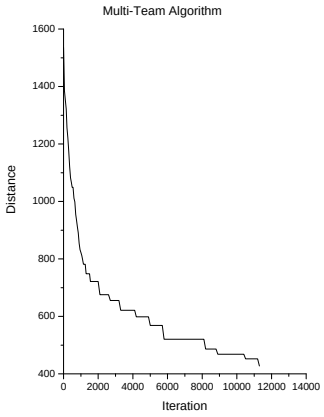


Figure 1. Experiment results for Eilon50 problem: (a)single agent (b)single-team algorithm (c)multi-team algorithm

Figure 2. Experiment results for KroA100 problem: (a)single agent (b)multi-agent systems

problem, the shortest distance of the multi-agent systems is 721.4. It is much shorter than that of single agent (1031.2). And it is also shorter than that of single agent with GA (841.2). As for KroA100 Problem, the shortest distance of the multi-agent systems is 23987.6, while the shortest distance of the single agent is 43819.7.

3.3 Comparison with other genetic methods

There are a lot of heuristics algorithms [10] for solving the TSP problem, such as the nearest neighbor algorithm, the insertion algorithm, the spanning tree algorithm, the 2-OPT exchange algorithm, and the 3-OPT exchange algorithm. The simulated annealing algorithm, genetic algorithms, the tabu search algorithm, and neural networks are also used to solve the TSP problem.

Whitley et al [14] used the genetic edge recombination operator to solve the TSP problem. They developed an edge map to construct an offspring that inherits as much information as possible from the parent structures. The edge map stores all the connections from two parents that lead into and out of a city.

Lin et al [7] incorporated genetic algorithms into the simulated annealing algorithm to solve the TSP problem. This approach is a simulated annealing algorithm with the population-based state transition and genetic-operator-based control.

Fogel [6] used a new genetic operator in genetic algorithms. The mutation operator will choose one part of cities and reverse the order of cities.

Ant Colony System(ACS) [4][3] is very effective in solving the TSP problem. In ACS, a set of cooperating agents cooperates to find the best solution to a problem. Agents used an indirect form of communication mediated by a pheromone they deposited.

The comparison of our system with other genetic algorithms is in the Table 1. Results using ACS are from [4], and results using EP are from [6]. We implemented the EP approach, but can not get the result as good as in [6]. Our results are shown as EP-1. For results using GA, results of Eilon50 are from [13] and KroA100 are from [2].

The ACS has the fastest speed among above methods. Our systems need to train two reinforcement learning algorithms (Q and CQ modules), while ACS only has one learning module similar to the reinforcement learning algorithm. Using neural networks to implement the reinforcement learning algorithm may be another reason for slower

Algorithm	Eilon50	KroA100
Single Team	425 [139,500]	21282 [2,925,000]
Multi-Team	425 [160,500]	21282 [4,125,000]
ACS	425 [1,830]	21282 [4,820]
GA	428 [25,000]	N/A [103,000)]
EP	426 [100,000]	N/A
EP-1	426 [350,000]	N/A
Optimum	425	21282

Table 1. Comparison of our system with other algorithms. The number in brackets is the number of tours searched.

learning speed of our system compared to that of ACS.

3.4 Discussions

Our approach extends existing work, in that it is not limited to bidding alone. For example, it is not just using bidding alone to form coalitions or bidding alone as the sole means for learning [1], and it is not a model of pure reinforcement learning [8]. Furthermore, it is not a pure evolutionary system [15]. It is the combination of the three aspects: reinforcement learning, evolution and bidding.

Although our program did not achieve the same level as ACS, it achieves a certain level starting from scratch. The cooperation among agents (through bidding) helps to simplify the learning process.

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 to 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 facilitate the learning in our approach.

4 Conclusions

In sum, in this work, we developed a bidding approach for establishing cooperation among a group of self-interested agents. The experiment shows our approach can achieve a certain level of performance in problem solving. The result of the experiments suggests that the bidding system may work well in general complex problems and domains.

ACKNOWLEDGMENT

This work was supported in part by ARI grant DASW01-00-K-0012, and a University of Missouri-Columbia startup fund.

References

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

[2] H. Bersini, C. Oury, and M. Dorigo. Hybridization of genetic algorithms. Technical Report IRIDIA/95-22, University of Bruxelles, 1995.

[3] M. Dorigo and G. D. Caro. Ant algorithms for discrete optimization. *Artificial Life*, 5(3):137–172, 1999.

[4] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

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

[6] D. B. Fogel. Applying evolutionary programming to selected traveling saleman problems. *Cybernetics and Systems: An International Journal*, 24:27–36, 1993.

[7] F.-T. Lin, C.-Y. Kao, and C.-C. Hsu. Applying the genetic approach to simulated annealing in solving some np-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1752–1767, 1993.

[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] G. Reinelt. TSPLIB. *http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/*.

[10] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, Berlin, Germany, 1991.

[11] R. Sun and C. Sessions. Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors. *IEEE Transactions on Systems, Man, and Cybernetics: Part B Cybernetics*, 30(3):403–418, 2000.

[12] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 16(3):185–202, 1992.

[13] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and travelling salesman: The genetic edge recombination operator. In *Proceedings of the third international conference in Genetic Algorithms*, pages 133–140, Palo Alto, CA, 1989. Morgan Kaufmann.

[14] D. Whitley, T. Starkweather, and D. Shaner. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In L. Davis, editor, *The Handbook of Genetic Algorithms*, pages 350–372. Van Nostrand Reinhold, 1991.

[15] 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.