Forming Coalitions for Breaking Deadlocks

Katsutoshi Hirayama* and **Jun'ichi Toyoda**
ISIR, Osaka University
8-1 Mihogaoka
Ibaraki, Osaka 567, JAPAN
{hirayama, toyoda}@ai.sanken.osaka-u.ac.jp

Abstract

When multiple agents solve their own problems while they interact with each other, it is helpful to form a *coalition*, which is a group of agents working together. Previous approaches to coalition formation have proposed to define the utility of coalitions and to use a strategy that agents form coalitions for getting higher utility. However, in some problems, the utility of coalitions is not easily obtainable because it might depend on various uncertain things. This paper describes a model of coalition formation where agents form coalitions for breaking deadlocks. In this model, agents solve Distributed Constraint Satisfaction Problems with an *iterative repair* method, and form coalitions when they get stuck at local minima. This model is suggested to realize a new approach to coalition formation. We also present problem solving strategies in coalitions: the *selfish* and the *altruistic*. These two strategies differ in the way to build a domain of variables. From our experimental results on *distributed 3-coloring problems*, the *altruistic group* performed better than the *selfish group*.

Introduction

Suppose that the environment where multiple agents have their own local problems; there are some interactions among their problems; and agents have a method to communicate with each other. In such environment, there may be many chances for cooperation among the agents to coordinate their problem solving. It might be one possibility for agents to cooperate with all other agents so that the cooperation will benefit all. That cooperation, however, seems to be expensive because the agents have to coordinate all their problem solving. Thus it is more likely that agents form a *coalition*, a subset of the agents working together, for benefiting only the members of the coalition.

*Corresponding author's current address: Faculty of Mercantile Marine Science, Kobe University of Mercantile Marine, 1-1, 5-chome, Fukae-Minami-machi, Higashi-Nada-ku, Kobe 658 JAPAN. E-mail address: hirayama@ti.kshosen.ac.jp

Although coalition formation is important strategy for problem solving in multi-agent systems, it is a difficult problem how agents should form coalitions. In previous approaches to this problem, they have proposed to define utility values that represent the benefits from forming coalitions, and to use a strategy that agents form coalitions for getting higher utility (Ketchpel 1994, Zlotkin & Rosenschein 1994). The utility values are given in advance so that problems can be solved in game-theoretic manner. However, in some problems, we consider that it is difficult to get a correct estimate of the actual benefit because it might depend on various uncertain things.

Our approach is different from the previous game-theoretic ones. It is very simple, and does not need to define utility values or something. The key idea is that agents should form coalitions when they come to deadlocks. We call the deadlock the state where there is no other way to improve the current state. Based on this idea, we have developed a model where agents solve Distributed CSP: Distributed Constraint Satisfaction Problems (Yokoo *et al.* 1992). In this model agents solve their local problems with an *iterative repair* method, which is a form of hill-climbing that reduces the number of constraint violations (Minton *et al.* 1992, Selman *et al.* 1992). Agents try to reduce the number of their *own* constraint violations, and some agents may consequently get stuck at *local minima*. Agents at local minima have no way to reduce the number of constraint violations. In the model agents form coalitions for breaking local minima. By forming coalitions, we mean that neighboring two agents in local minima contract to make the constraints between them consistent. Forming coalitions in this way leads to *filling in* local minima, i.e., preventing agents from getting stuck at the same local minima repeatedly.

In this paper we describe the coalition formation model in detail, and present two problem solving strategies used in coalitions. One is the *selfish*. After forming a coalition, the selfish coalition assigns its variables a set of values that may violate many constraints being shared with others. The other strategy is the *altruistic*. This coalition assigns its variables a

set of values that causes the smallest number of constraint violations. While the selfish coalition can easily get new assignments because all it has to do is to search for one set of values that satisfies its local constraints, the altruistic coalition has to have a hard time searching for all possible sets of values. For evaluating these strategies, we tested the *selfish group* and the *altruistic group* on *distributed 3-coloring problems*. The results show that the altruistic group worked better than the selfish group.

This paper is organized as follows. To begin with we provide a detail of the model. Next we present the two strategies used in coalitions, and furthermore show experimental results for evaluating these strategies. Then we also touch previous distributed constraint satisfaction algorithms, and point out the difference between our algorithm and previous ones. The final part of this paper contains conclusions and future work.

The Model

This section describes the model using examples which enable understanding the detail of it.

Distributed CSP

CSP can be stated as follows (Kumar 1992): We are given a set of variables, a finite and discrete domain for each variable, and a set of constraints. Each constraint is defined over some subset of the variables. The goal is to find one set of (or all sets of) assignments to the variables such that the assignments satisfy all the constraints.

In Distributed CSP each agent has variables, domains, and constraints. The goal of each agent is to find one set of assignments to its variables that satisfy all the constraints. The constraint, however, is usually defined over a set of variables including other agents' variables. Accordingly, agents must communicate each other to achieve their goals. We introduce the following specifications of a communication model that can be seen in (Yokoo *et al.* 1992):

- Communication between agents is done by sending messages;
- One agent can send messages to another *iff* one of the former's constraints is defined over a set of variables including the latter's variables (we refer the latter agent as the former's *neighbor*);
- The delay in delivering a message is finite;
- Messages are received in the order in which they were sent.

On the other hand, we assume the following without loss of generality (Yokoo *et al.* 1992):

- Each agent has exactly one variable;
- Each constraint is binary, that is, it is defined over two variables;

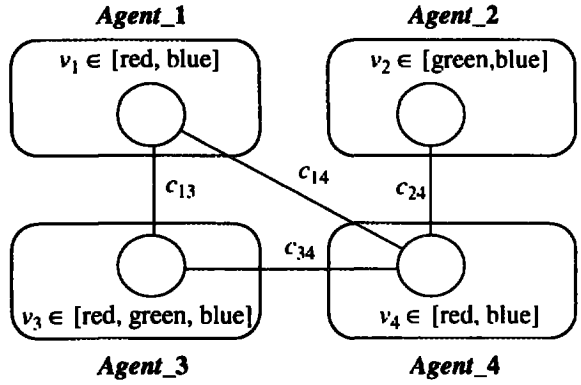

Figure 1: A distributed graph coloring problem. v_i is a variable, and c_{ij} is a constraint that prohibits from assigning the same value to v_i and v_j. This figure indicates that, for example, *Agent_1* has v_1 as a variable, [red, blue] as a domain, and c_{13} and c_{14} as constraints.

- Each agent has all constraints relevant to its variable.

In the above definition, we should notice that there is no global control and no agent knows the entire problem. In Figure 1 we show a typical example of Distributed CSP: a *distributed graph coloring problem*. A graph coloring problem consists of a graph and the requirement to find a color for each node (from a set of colors) such that no pair of adjacent nodes has the same color. Nodes and arcs correspond to variables and constraints, respectively. A distributed graph coloring problem can be considered to be a graph coloring problem where the nodes in a graph are distributed among agents.

An Iterative Repair Method for Solving Distributed CSP

A number of papers have presented algorithms using an *iterative repair* method for solving CSP (Minton *et al.* 1992, Selman *et al.* 1992, Morris 1993, Selman & Kautz 1993). This method is a kind of local search, which searches local area of problem space for a better solution, and makes local changes of assignments repeatedly. It works as follows:

1. Generate an initial "solution" containing constraint violations;
2. Until the current "solution" is acceptable, improve it by making local changes that reduce the total number of constraint violations.

While this method is very simple, it has made great success on some problems, for instance, n-queens problems, graph coloring problems, and propositional satisfiability problems.

In this subsection we present an iterative repair method for solving Distributed CSP. The outline of this

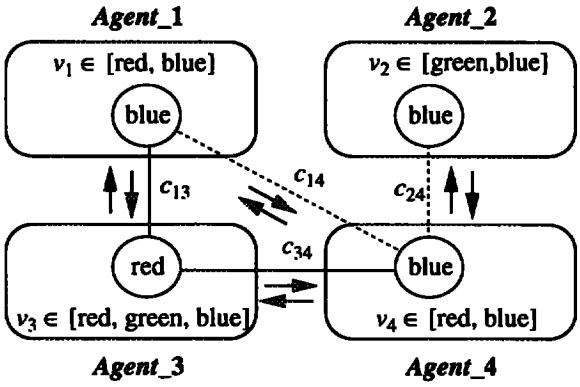

Figure 2: Informing neighbors about current assignments. Each agent informs its neighbors about the current assignment by sending messages. This figure shows that, for example, *Agent_1* sends messages to *Agent_3* and *Agent_4* which indicate that "v_1 is blue". Note that a dotted line shows a constraint violation, and an arrow shows message passing between agents.

method is that agents repeatedly make local changes to reduce the numbers of their *own* constraint violations while they mutually exclude neighbors' changes through negotiations. Using an example in Figure 2 – 7 we explain how this method proceeds.

Informing neighbors about current assignments. Agents pick out any values as initial assignments, and inform their neighbors by sending messages. When the messages are received, agents construct *agent_views*, each of which is a list of the pair: [neighbor's identifier, neighbor's current assignment].

In Figure 2, *Agent_1*'s agent_view is as follows: [[*Agent_3*, v_3 = red], [*Agent_4*, v_4 = blue]].

Negotiations before local changes. Agents with constraint violations start negotiations for local changes by sending the current and smallest numbers of constraint violations. Agents will make local changes *iff* all their neighbors approve of the changes. One agent's (we call the agent A_1) change is approved by the other (we call the agent A_2) in the case that: (i) All of A_2's constraints are satisfied or (ii) A_1's change will reduce more number of constraint violations than A_2's will. Comparing their identifiers breaks ties (i.e., when both agents' changes reduce the same, A_1's change is approved if A_1's identifier precedes A_2's in the alphabetical order).

In Figure 3 and 4, *Agent_1*, *Agent_2* and *Agent_4* start negotiations with their neighbors, and only *Agent_2* is approved by all its neighbors. Consequently, *Agent_2* changes its assignment into green as in Figure 5. To reduce the number of messages, we have made an additional condition for the agents that start negotiations in the actual implementation.

Forming Coalitions

While one agent's local change has a good effect on its *own* problem solving (in the sense that it reduces the number of constraint violations), it may have bad effects on other agents'. See Figure 5 for example. This figure represents the state after *Agent_2*'s local change. This change forces *Agent_1* and *Agent_4* to be in the state where no local change reduces the current number of constraint violations. We call the state a *local minimum*. *Agent_1* and *Agent_4* get stuck at local minima in Figure 5.

For breaking local minima, agents form coalitions in our model. By forming coalitions, we mean that agents contract to make the constraints among them consistent. Neighboring two agents form a coalition when (i) they are in local minima and (ii) they share a violated constraint. Once some agents form a coalition, they continue to perform iterative repair while keeping their local constraints consistent. We will give details of agents' behavior in a coalition in the next section.

For example in Figure 5, 6 and 7, *Agent_1* and *Agent_4* form a coalition through negotiations for the right to be the *manager*: the agent which organizes their local problem solving (We will detail in the next section). After that, the coalition continues to perform iterative repair, keeping the constraint c_{14} consistent.

We should notice that forming coalitions like this has an important effect. Coalitions, as mentioned above, keep some constraints consistent once they are violated at local minima. That implies the fact that agents *fill in* local minima by forming coalitions. In other words, forming coalitions prevents agents from getting stuck at the same local minima repeatedly.

Other Models

Previous studies on coalition formation assume that agents estimate utility values that they will receive from membership in coalitions, and they form coalitions for getting higher utility (Ketchpel 1994, Zlotkin & Rosenschein 1994). This, however, drives us to the question how agents get correct estimates of the utility of coalitions in advance. We consider that it is hard to obtain those estimates for the problems like Distributed CSP, because the costs of problem solving by agents or coalitions are greatly dependent on others' current and future assignments. Our model suggests a new approach to coalition formation where agents form coalitions for breaking deadlocks instead of for getting higher utility.

Organization self-design in (Ishida *et al.* 1990) proposes two operator: *composition* and *decomposition*. Composition is an operation for forming groups, aiming at saving hardware resources at the cost of the benefit from parallel execution. While this model uses organizational statistics that represents the behavior of the organization as a whole, agents in our model form groups using no such global information.

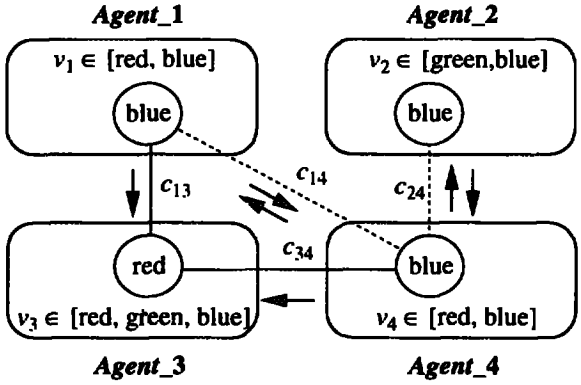

Figure 3: Negotiations for local changes. By sending the current and smallest numbers of constraint violations, agents (with constraint violations) start negotiations for local changes. For example, *Agent_2* sends a message to *Agent_4* which indicates that "# of current violations = 1 and # of minimum violations = 0" in this figure. *Agent_3* has no constraint violation, hence sending no messages.

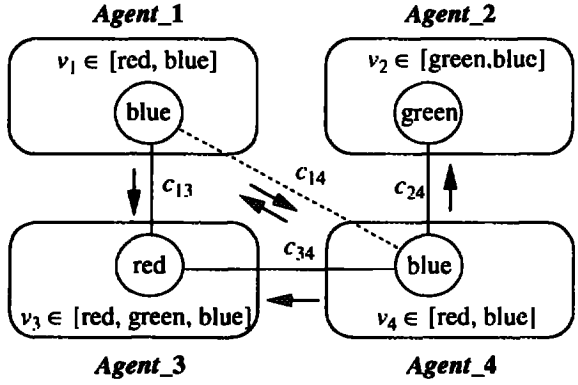

Figure 5: Negotiations for the right to be the manager. The agents form a coalition when they are in local minima and share a violated constraint. This figure shows that *Agent_1* and *Agent_4* are in local minima and share a violated constraint c_{14}. They start negotiations for the right to be the manager of a coalition by sending messages which indicate that: "# of current violations = 1 and # of minimum violations = 1".

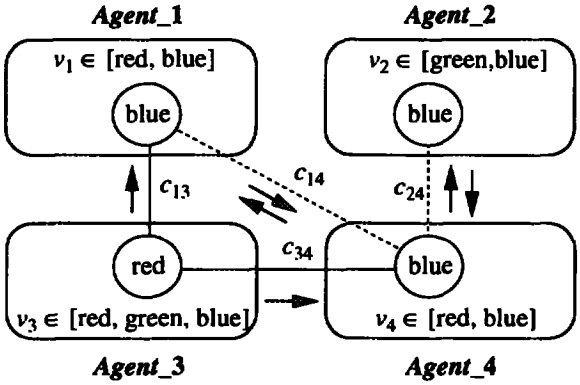

Figure 4: Results of negotiations for local changes. An agent compares its current and smallest numbers of constraint violations with the sender's, and decides on whether to approve of the sender's change. In this figure, *Agent_1* approves of *Agent_4*'s change, whereas *Agent_2* does not. *Agent_3* approves of both *Agent_1*'s and *Agent_4*'s, and *Agent_4* does only of *Agent_2*'s.

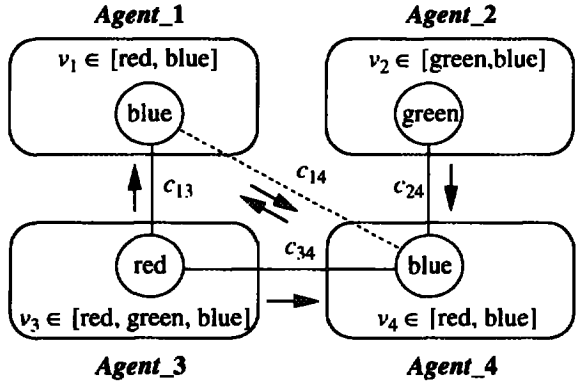

Figure 6: Results of negotiations for the right to be the manager. An Agent decides on whether to approve of the sender's right to be the manager in the same manner used in negotiations for local changes. In this figure, *Agent_1* does not approve of *Agent_4*'s right, whereas *Agent_2* does. *Agent_3* approves of both *Agent_1*'s and *Agent_4*'s, and *Agent_4* does of *Agent_1*'s.

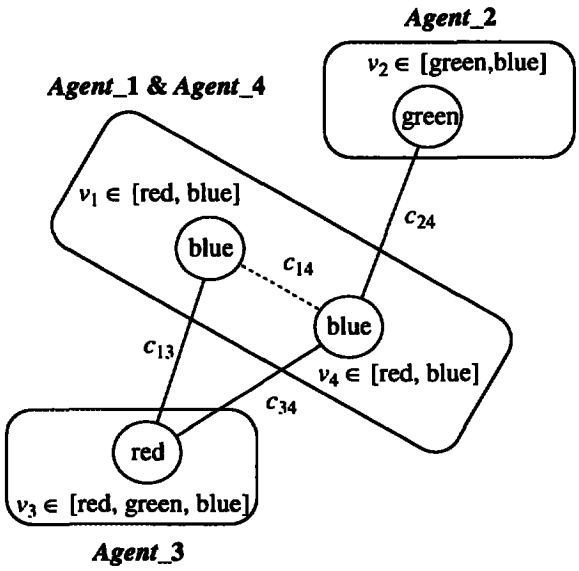

Figure 7: Forming a coalition. An agent becomes the manager of a coalition if all its neighbors approve of that. In this figure, *Agent_1* becomes the manager of a coalition, which consists of *Agent_1* and *Agent_4*, with its neighbors' approval.

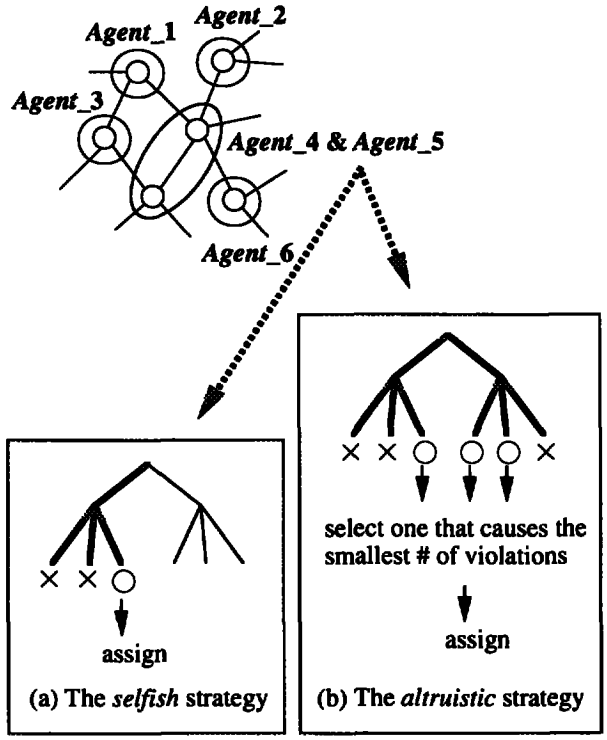

Figure 8: The *selfish* and *altruistic* strategies. This figure illustrates the initial behaviors of the selfish and altruistic managers. Note that the initial behavior is the one which is performed until the manager finds initial assignments.

Strategies in Coalitions

As describe earlier, members of a coalition are in the contract that they make their local constraints consistent. That is, they have to make local changes such that their new assignments do not violate their local constraints. It is a next problem how the members realize that. In this paper, we assume a simple form of problem solving: the manager gathers all variables, domains, and constraints in a coalition, and solves their local CSP using various strategies. The purpose of this section is to present two strategies for the manager: the *selfish* and the *altruistic*.

Strategy 1: the selfish

Right after forming a coalition, the selfish manager searches for one set of possible values, and assigns it to variables. The cost of the search is relatively low since it only has to get one set of values that satisfies constraints among the variables. It, however, has no consideration for the neighbors of the coalition because that set of assignments may violate many constraints with the neighbors. This strategy will succeed when neighbors are able to repair constraint violations immediately. When not, the manager is required to search for any other sets of values. All sets of values that have been obtained are kept as a new domain of the variables.

Using a new domain, the manager performs iterative repair such as follows:

1. it counts the current number of constraint violations, and searches for one set of values that reduces the number of violations in the current domain;
2. if such a set exists, it starts a negotiation for a local change, and if not, it searches for a new set that reduces the number of violations in the unsearched problem space instead of the current domain;
3. when such a new set is found, it negotiates for a local change, and when not it does for forming a new coalition (for the right to be the manager).

In short the manager builds a new domain of variables on demand, performing iterative repair.

Strategy 2: the altruistic

The altruistic manager is more self-sacrificing. It searches for all possible sets of values to variables, selects one set that violates the smallest number of constraints with neighbors, and assigns it to the variables. It has a hard time to search for all possible sets so that the assignments may cause minimum constraint violations. We consider that this strategy works well when the size of local CSP is relatively small.

	the *selfish group*					the *altruistic group*				
n	step	check	message	max	success	step	check	message	max	success
10	59.5	639.1	507.3	2.50	100%	52.2	554.6	446.7	2.18	100%
20	136.8	7133.5	1795.1	6.86	100%	114.4	4956.7	1406.3	6.25	100%
30	210.5	68374.3	3769.1	11.24	100%	172.2	49207.2	2796.9	10.12	100%
40	271.8	411162.6	6205.4	16.65	77%	214.3	190506.9	4435.4	13.98	95%
50	304.8	530180.3	8465.0	17.49	58%	236.6	437372.4	5962.7	15.77	75%

Table 1: Experimental results on distributed 3-coloring problems

It is simple for the manager to perform iterative repair after forming a coalition. The manager searches for all sets of values, and builds a complete domain of variables before performing iterative repair (note: the selfish manager builds a new domain on demand). With this complete domain, the manager performs iterative repair in the same way as it does before forming the coalition.

We show the initial behaviors of the selfish and altruistic managers in Figure 8.

Experiments

We assume two groups of agents: the *selfish group* and the *altruistic group*, each of which uses an uniform strategy. The selfish group consists of the agents that use the selfish strategy, and on the other hand, the altruistic group is a group of agents using the altruistic strategy. In this section, we present experimental results on the performance of the two groups when they solved *distributed 3-coloring problems*. *3-coloring problems* are graph coloring problems where we need to color each node using 3 different colors. As reported in (Adorf *et al.* 1990, Minton *et al.* 1992), an iterative repair method results in poor performance for sparsely-connected graphs of 3-coloring problems. The sparsely-connected graph consists of m arcs and n nodes, where $m = 2n$. We expect that an iterative repair method will perform poorly (and agents will form coalitions repeatedly) for those graphs of distributed 3-coloring problems. Hence we applied the selfish and the altruistic groups to the sparsely-connected graphs, varying n from 10 to 50. For each n, we generated 10 problems using the method described in (Minton *et al.* 1992), each of which has 10 trials with randomly generated initial values.

These experiments were made on the simulator, which is developed for realizing concurrent activity among agents. All settings of this simulator are same as the one in (Yokoo 1993) in which each agent maintains its own simulated clock and sets it forward by one simulated time unit whenever it performs one cycle of computation (one cycle consists of reading all messages, performing local computation, and sending messages); a message issued at time t is available to the recipient at time $t + 1$. For each trial, we measured the following required to reach the state where all agents satisfy their constraints:

step : simulated time;

check : the total number of constraint checks;

message : the total number of messages.

At the time all agents satisfy their constraints, we also measured:

max : max size of coalitions.

A bound of a million number of constraint checks was employed so that the experiments could be finished in a reasonable amount of time. For each n, we measured:

success : the percentage of trials finished within the bound.

If a group did not find a solution within the bound, it stoped the trial, and kept the above data at that time. The results are shown in Table 1. All figures (except ones in success column) are averaged over 100 trials for each n.

For all cases, the selfish group consumed more steps, checks, and messages. As stated in the previous section, the selfish strategy could be efficient if neighbors repaired constraint violations immediately. That corresponds to the case that agents fall in "shallow" local minima. The sparsely-connected graphs, however, have many "deep" local minima, and hence the selfish group did not work well.

For large n, both groups formed very large coalitions. The size of local CSP increases with coalition size, and the altruistic group may hence be inefficient. However why did the selfish group perform so poorly for large n? We should notice that agents formed larger coalitions in the selfish group than in the altruistic group. The selfish strategy may violate more constraints, and

it is likely that these extra constraint violations cause extra coalition formation. The selfish group can be inefficient if the cost of extra coalition formation exceeds the saving of the search cost in coalitions. It seems to be for this reason that the selfish group showed poor performance also for large n.

Distributed Constraint Satisfaction Algorithms

We have considered our model to be a new approach to coalition formation. In this section, we will focus on another aspect of the model: a distributed constraint satisfaction algorithm.

Previous Algorithms

The purpose of distributed constraint satisfaction algorithms is to solve Distributed CSP efficiently. As distributed constraint satisfaction algorithms, Yokoo has proposed two algorithms, which are called *asynchronous backtracking* (Yokoo *et al.* 1992) and *asynchronous weak-commitment search* (Yokoo 1993), respectively.

Asynchronous backtracking is a complete algorithm in which agents change their assignments asynchronously and concurrently while they exchange *ok?* and *nogood* messages. (Note that a distributed constraint satisfaction algorithm is complete when following conditions are fulfilled: agents eventually find their solutions if there is at least one set of solutions; and agents stop their process if there exsists no set of solutions) By sending ok? messages, agents inform their neighbors about their own current assignments. When an agent receives an ok? message, it proceeds as follows:

1. updates its agent_view;
2. checks whether its current assignment is consistent with the agent_view. The assignment is consistent when satisfying all constraints being shared with agents with higher priorities. Priority values are defined by using the alphabetical order of agents' identifiers. An agent with a preceding identifier in the alphabetical order has higher priority;
3. if the current assignment is not consistent, selects a new value which is consistent, assigns the value, and sends ok? messages to neighbors. If the agent cannot find such a new value, it makes a *nogood*, which is a set of values that cannot be a part of any final solutions, and sends the nogood as nogood messages to neighbors. That corresponds to backtracking.

When an agent receives a nogood message, on the other hand, the agent records the nogood as a new constraint, changes its assignment to the value which is consistent with the agent_view, and sends ok? messages to its neighbors. If the agent cannot find a consistent value, it makes another nogood to send its neighbors.

Although asynchronous backtraking is the first distributed constraint satisfaction algorithm, it is not so efficient. For the increase of efficiency, Yokoo has introduced two heuristics into asynchronous backtracking, and called the resultant algorithm asynchronous weak-commitment search. One of the introduced heuristics is min-conflict heuristic (Minton *et al.* 1992): when selecting a set of new values, an agent selects the one that minimizes the number of conflicts. Another heuristic is to change priority values of agents dynamically whenever backtracking is occurred. According to experimental results on distributed n-queens problems and network resource allocation problems, it has been proved that these two heuristics dramatically improve the perforance of asynchronous backtracking.

The Feature of Our Algorithm

In the previous algorithms agents change their assignments asynchronously and concurrently, that is, they do as soon as they find new values which are consistent with their agent_views. Consequently, neighboring agents often change their assignments simultaneously without knowing each other's changes. We expect that this property may have a bad influence on efficiency. The algorithm which we have developed, on the other hand, uses the negotiation procedure which realizes that neighboring agents mutually exclude their changes. Thereby it seems reasonable to suppose that our algorithm may reduce the total number of assignment changes. It, however, is debatable how many extra messages the negotiation procedure requires. We ought to compare our algorithm with previous ones in analytical/experimental manner as future work.

Conclusions

We described a new coalition formation model where agents form coalitions for breaking deadlocks, and presented two strategies: the selfish and the altruistic, for coalitions to solve their local problems. We then made an experiment on the performance of the strategies, and obtained the results which indicate that the selfish strategy causes extra coalition formation, and thereby performing poorly on distributed 3-coloring problems.

Finally, we give several possibilities of future work: investigate when to break off coalitions; investigate other strategies of coalitions; and investigate the performance of a group with heterogeneous strategies.

References

Adorf, H. M., and Johnston, M. D. 1990. A Discrete Stochastic Neural Network Algorithm for Constraint Satisfaction Problems. In *Proceedings of the International Joint Conference on Neural Networks*, (III)917–924.

Ishida, T.; Yokoo, M.; and Gasser, L. 1990. An Organizational Approach to Adaptive Production Systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 52–58.

Ketchpel, S. 1994. Forming Coalitions in the Face of Uncertain Rewards. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 414–419.

Kumar, V. 1992. Algorithms for Constraint Satisfaction Problems: A Survey. *AI magazine* 13(1): 32–44.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence* 58: 161 205.

Morris, P. 1993. The Breakout Method for Escaping from Local Minima. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 40–45.

Selman, B.; Levesque, H.; and Mitchell, D. 1992. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 440–446.

Selman, B., and Kautz, H. 1993. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 290–295.

Yokoo, M.; Durfee. E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In *Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems*, 614–621.

Yokoo, M. 1993. Dynamic Variable/Value Ordering Heuristics for Solving Large-Scale Distributed Constraint Satisfaction Problems. In *Proceedings of the Twelfth International Workshop on Distributed Artificial Intelligence*.

Zlotkin, G., and Rosenschein, J. S. 1994. Coalition, Cryptography, and Stability: Mechanisms for Coalition Formation in Task Oriented Domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 432–437.