CSE-TR-101-91: The University of Michigan

Distributed Constraint Optimization as a Formal Model of Partially Adversarial Cooperation

Makoto Yokoo*and Edmund H. Durfee

Dept. of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109
yokoo/durfee@caen.engin.umich.edu

Abstract

In this paper, we argue that partially adversarial and partially cooperative (PARC) problems in distributed artificial intelligence can be mapped into a formalism called distributed constraint optimization problems (DCOPs), which generalize distributed constraint satisfaction problems [Yokoo, *et al.* 90] by introducing weak constraints (preferences). We discuss several solution criteria for DCOP and clarify the relation between these criteria and different levels of agent rationality [Rosenschein and Genesereth 85], and show the algorithms for solving DCOPs in which agents incrementally exchange only necessary information to converge on a mutually satisfiable solution.

1 Introduction

In building a taxonomy of the field, researchers in Distributed Artificial Intelligence (DAI) have divided DAI research into two camps [Bond and Gasser 88]. One camp is distributed problem solving, which is concerned with how to get agents that are working on common problems to cooperate effectively. The mutual dependencies between the agents and the sharing of goals between them have led some researchers to characterize distributed problem-solving agents as *benevolent* [Rosenschein and Genesereth 85]. The other camp involves multiagent systems, in which autonomous, *self-interested* agents who might have completely different goals still have to coordinate their activities within their shared environment.

*Visitig from NTT Communication Science Laboratories, Sanpeidani Inuidani, Seika-cho Soraku-gun, Kyoto 619-02 Japan, e-mail: yokoo%cslab.kecl.ntt.jp@relay.cs.net

While this characterization helps distinguish extreme ends of a broad spectrum of DAI problems, the truth is that most problems of interest fall in the grey area between these extremes. For example, although the vehicle tracking agents in the Distributed Vehicle Monitoring Testbed [Lesser and Corkill 83] lean towards cooperation because they share a common goal of mapping overall vehicle movements, each operationalizes the common goal differently because of its own local knowledge and data. As a result, the cooperative agents tend to value their own partial results most highly, leading to exchanges of distracting information and competition in getting other agents to extend alternative partial solutions. Thus, the "benevolent" agents in the distributed problem-solving task are partially adversarial. On the other hand, self-interested and even competitive agents in a multiagent system often have to be partially cooperative, such as when two competing sports teams must cooperate to arrange a game. Similarly, in delivery domains studied in DAI [Durfee and Montgomery 90] the agents that are working on separate delivery tasks must still cooperate to satisfy constraints imposed by the environment (such as the constraint that they cannot occupy the same place at the same time). Thus, self-interested and adversarial agents often must also be partially cooperative.

A simple example of a partially-adversarial/partially-cooperative (PAPC) problem is shown in Figure 1. This problem is called the *lazy* 4 queens problem, where 4 agents exist, each of which tries to place its queen on the assigned column so that the queens do not threaten each other. Furthermore, each queen has its initial position, and each agent prefers moving as little as possible from its initial position. In this example, the existence of hard constraints (that queens do not threaten each other) forces agents to cooperate since acting independently will likely lead to a poor outcome. At the same time, they are partially adversarial because of their different preferences (weak constraints) which generally cannot be maximally satisfied simultaneously. This PAPC problem captures the essence of many more difficult problems, including the adversarial cooperation between the president and the congress, and the grudging cooperation between labor unions and management [Sycara 85].

Rosenschein and his colleagues have extensively studied how agents can select *rational* actions in PAPC situations [Rosenschein and Genesereth 85]. In their work, each agent makes an offer, which consists of a combination of actions of all agents. The agents select a mutually satisfiable solution from the intersection of all of their sets of offers. Rosenschein concentrates on defining different levels of rationality of the agents in deciding on what offers to make, and how different rationality levels impact the ability of the agents to arrive at solutions. A limitation of this approach is that an agent is assumed to know outcomes of all agents for all possible combinations of actions in advance (that is, it knows the whole payoff matrix). This assumption becomes impractical when the number of agents and the number of values increase. Furthermore, in adversarial domains, getting agents to truthfully reveal their payoffs is problematic.

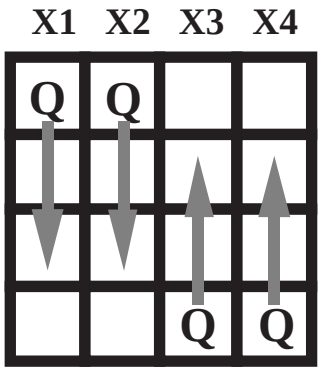

Figure 1: Lazy 4 queens Problem

In this paper, we argue that PAPC problems can be mapped into a formalism called distributed constraint optimization. Distributed constraint optimization problems (DCOPs) generalize distributed constraint satisfaction problems [Yokoo, *et al.* 90] by varying degrees of satisfaction for a preference (weak constraint), as well as insisting that hard constraints be fully satisfied. Thus, agents solving DCOPs can be adversarial across different potential solutions that satisfy hard constraints. Whether the agents can eventually converge on a mutually satisfiable solution depends on the criteria that they use for evaluating preference satisfactions, which is related to different levels of agent rationality [Rosenschein and Genesereth 85]. We have developed new algorithms for solving DCOPs in which agents incrementally exchange only necessary information to converge on a mutually satisfiable solution.

2 Distributed Constraint Optimization Problem

In this paper, we use the following simplified definition of a distributed constraint optimization problem. We assume all weak constraints are aggregated by one objective function for each agent. Without loss of generality, an agent is assumed to have exactly one variable, and an objective function is defined on the assignment of all variables. These assumptions can be relaxed straightforwardly to the case that one agent has several variables, or the objective function of an agent is defined on a subset of the assignment of all variables.

- There exist agents $1, \ldots, n$.

- Agent i has exactly one variable x_i, which takes its value from a finite domain D_i.
- A candidate solution S is an assignment of values to all variables $\{(x_1, d_1), \ldots, (x_n, d_n)\}$.
- There exist a set of (hard) constraints, where a constraint $P_k(x_{k1}, \ldots, x_{kj})$ is a predicate which is defined on the Cartesian product $D_{k1} \times \ldots \times D_{kj}$. This predicate is true iff the instantiations of these variables are compatible with each other.
- Agent i has its own objective function F_i, which takes S as an argument and returns a cost value.

For example, the lazy 4 queens problem can be formalized as follows:

- There exist agents 1,2,3,4, with variables x_1, x_2, x_3, x_4, respectively, where the domain of each variable is $\{1, 2, 3, 4\}$.
- There exists a constraint P_{ij} between x_i and x_j, which is satisfied iff $(x_i \neq x_j) \wedge (|x_i - x_j| \neq |i - j|)$.
- Agent i has an objective function F_i, which returns $|x_i -$ initial-position$_i|$

A solution of a DCOP must satisfy all hard constraints. Generally, several such solutions exist, in which case each agent prefers solutions with smaller values of its objective function (lower costs). Since the values of all objective functions might not be minimized by the same solution, as in the lazy 4 queens example, the agents must agree on a mutually satisfiable solution, rather than on a globally optimal solution. A mutually satisfiable solution strikes a balance on the agents' preferences, and how the balance is struck depends on the solution criterion they use.

3 Solution criteria for DCOP

With the existence of multiple objective functions, solution criteria depend on the goal of the problem solving. In this section, we show several different criteria.

3.1 Pareto Optimal Solution

A solution S is a *pareto optimal* solution iff

$$\neg\exists S'(\exists i(F_i(S') < F_i(S)) \wedge \forall j(F_j(S') \leq F_j(S)))$$

A pareto optimal solution is a solution that no other solution will improve upon with respect to *all* objective functions, in other words, no other solution *dominates* it.

Although pareto optimality is a desirable characteristic for a mutually satisfiable solution, there can be multiple pareto optimal solutions. In Figure 1, there

exist two solutions, i.e., $\{(x_1,2),(x_2,4),(x_3,1),(x_4,3)\}$, where the costs to the agents are 1, 3, 3, 1, respectively, and $\{(x_1,3),(x_2,1),(x_3,4),(x_4,2)\}$, where the costs are 2, 0, 0, 2, respectively. Both solutions are pareto optimal, but agents 1 and 4 prefer the first solution while agents 2 and 3 prefer the second solution. One way for selecting a solution among multiple pareto optimal solutions is coin flipping. However, agents have to negotiate over the probability weighting of the "coin." For example, if agents agree on a coin giving a probability of 0.5 for each solution, the expected cost will be 1.5 for all agents.

3.2 Weighted Sum Optimal Solution

Suppose the importance of agent i's objective function F_i is represented by a weight W_i. A solution S is a weighted sum optimal solution iff

$$\neg\exists S'(\sum_{i=1,\ldots,n} W_i \cdot F_i(S') < \sum_{i=1,\ldots,n} W_i \cdot F_i(S))$$

This criterion will be appropriate for the case that agents have a common goal, and the degree of achievement of the common goal can be represented by the weighted sum of objective functions. In centralized, non-distributed optimization problems, or in purely benevolent distributed problem-solving networks, we can assume the existence of such a global optimal criterion. However, if agents are independent, we cannot assume the existence of such a commonly-held global criterion, because each agent will weight its own preferences above those of the other agents. If the agents can agree, in fairness, to weight their objective functions equally, then the set of weighted sum optimal solutions they derive is simply a subset of the pareto optimal solutions. Even in this case, the agents need some sort of mechanisms to ensure that they truthfully report the degree to which a solution satisfies their objective functions, and to ensure that agents whose preferences are sacrificed for the larger good abide by the decision.

3.3 Worst Case Optimal Solution

A solution S is a worst case optimal solution iff

$$\neg\exists S'(\max_{i=1,\ldots,n} F_i(S') < \max_{i=1,\ldots,n} F_i(S))$$

This definition means that the cost of the worst agent will be smaller than any other possible solutions. A worst case optimal solution is not necessarily a pareto optimal solution, but at least one worst case optimal solution is also a pareto optimal solution, since a solution which dominates a worst case optimal solution is also a worst case optimal solution.

If agents have a common goal, and the degree of the achievement of the common goal is determined by the worst value of the objective functions, then the worst case optimal solution is appropriate. For example, if the value of the

objective function of each agent represents the required time for solving its local problem, and the common goal is achieved when all agents have solved their local problems (e.g., the pursuit problem [Benda 86], distributed scheduling problem [Sycara, *et al.* 90]) the worst case optimal solution minimizes the required time for achieving the common goal.

When agents are independent, a worst case optimal solution can be a mutually satisfiable solution since it is a fair solution. In [Rosenschein and Genesereth 85], *similar bargainers assumption* is introduced to represent the equality between agents. Under the similar bargainers assumption, an agent i will only offer solutions with costs lower than N iff another agent j will also only offer solutions with costs lower than N. For example, if agent i is not willing to accept a solution S_i where $F_i(S_i) = N$, then agent j will also refuse to accept a solution S_j where $F_j(S_j) = N$. This assumption means that agents have the same threshold value for accepting and not accepting solutions.

To reach deals in Rosenschein's formalization, agents must include worst case optimal solutions in their offers, otherwise agents will fail to reach a mutual agreement[1]. This fact can be explained as follows. Suppose an agent j does not offer a worst case optimal solution S, where $F_j(S) = N'$ and $\max_{i=1,\ldots,n} F_i(S) = N$, since agent j wants solutions where F_j will be less than N'. By the similar bargainers assumption, other agents also want solutions costing them less than N'. By the definition of worst case optimal solutions, no solution S' will satisfy the condition $\max_{i=1,\ldots,n} F_i(S') < N' \leq N$. Therefore, there is no solution which satisfies the requirements of all agents, and agents will fail to reach a mutual agreement. In other words, if all agents insist on lower costs than N, there is no room for a mutual agreement. In Figure 1, S=$\{(x_1,3),(x_2,1),(x_3,4),(x_4,2)\}$, where the costs are 2, 0, 0, 2, is a worst case optimal solution. Suppose agent 1 does not like this solution, since agent 1 wants solutions with lower costs than 2. By the similar bargainers assumption, all agents want solutions with costs lower than 2, no agreement can be reached.

4 Algorithms for DCOP

In this section, the algorithms for solving DCOP are described. We first show the algorithms for finding a worst case optimal solution, then describe how the algorithms for the other two criteria can be obtained by modifying these algorithms.

The main idea for solving DCOP is to establish (and revise) a threshold value for the objective functions, thus turning them into boolean predicates whose satisfaction depends on the current threshold. If all constraints are boolean predicates, the asynchronous backtracking algorithm [Yokoo, *et al.* 90] can find a solution which satisfies all constraints, or find that there is no such solution.

[1] More precisely, they must include the worst case optimal solutions which are also pareto optimal solutions.

By changing the threshold value and iteratively solving the DCSPs, we can find a solution which optimizes the objective functions under given criteria. Basically, there are two ways of changing the threshold.

- **Distributed Depth-first Branch & Bound (DDBB):**
  Set the initial threshold to some upper-bound and solve the corresponding DCSP; then keep on decreasing the threshold and solving the corresponding DCSP until no solution is possible. Return the most recent solution.
- **Distributed Iterative Deepening (DID):**
  Set the initial threshold to some lower-bound and attempt to solve the corresponding DCSP; then keep on increasing the threshold and attempting to solve the corresponding DCSP until a solution is found within the threshold. Return the first such solution.

These two algorithms correspond to two typical algorithms for solving centralized, non-distributed optimization problems, i.e., the depth-first branch & bound algorithm, and the best-first branch & bound algorithm, such as the A* algorithm [Kumar 87]. Specifically, the DID algorithm can be considered as a distributed version of the iterative deepening A* algorithm [Korf 85].

4.1 Distributed depth-first branch & bound algorithm (DDBB)

Each agent has a threshold value c for its objective function F_i, and treats $F_i(S) < c$ as a boolean constraint [2]. Initially, agents set this threshold value to some upper-bound or ∞. The DDBB algorithm is described as follows.

1. Perform the asynchronous backtracking.
2. If a solution S is found, then store S and find the maximal cost value $c_{max} = \max_{i=1,\ldots,n} F_i(S)$, set c to c_{max}, goto 1.
3. If there is no solution, return the most recently stored solution, terminate the algorithm.

For example, to solve the problem in Figure 1, agents first solve the 4-queens problem without considering the objective functions. Suppose a solution $\{(x_1,2),(x_2,4),\ (x_3,1),(x_4,3)\}$ is found by using the asynchronous backtracking algorithm. The cost value of each agent is 1, 3, 3, 1, respectively. Therefore, the maximal cost is 3. Agents set their threshold to 3 and try to find a solution where the costs are less than 3. Then, agents find a solution $\{(x_1,3),(x_2,1),(x_3,4),(x_4,2)\}$, in which costs for each agent are 2, 0, 0, 2, respectively. Agents then set their threshold to 2 and try again to find a solution. However, there is no solution better than this threshold. Therefore, agents can guarantee that $\{(x_1,3),(x_2,1),(x_3,4),(x_4,2)\}$ is a worst case optimal solution.

[2] We can assume that agent i has its own heuristic function H_i that takes any subset of assignments $\{(x_1,d_1),\ldots,(x_n,d_n)\}$, and returns the estimated value of the objective function. This heuristic function can be used for early pruning in asynchronous backtracking

4.2 Distributed iterative deepening algorithm (DID)

In the DID algorithm, agents initially set their threshold cost value c to some lower-bound or 0, and treat $F_i(S) \leq c$ as a boolean constraint. The original asynchronous backtracking algorithm is modified so that when there is no solution, the algorithm returns the new lower-bound of the values of the objective functions. The modification is described as follows.

In the asynchronous backtracking algorithm, agents exchange *nogoods* with each other. A nogood is a set of value assignment to variables which does not satisfy constraints. For example, in the problem of Figure 1, if $x_1 = 1$ and $x_2 = 3$, there is no value for x_3 which satisfies constraints. Therefore, $\{(x_1, 1), (x_2, 3)\}$ is a nogood. When an empty nogood is found, it means that there is no solution which satisfies all constraints and the asynchronous backtracking algorithm terminates. In the DID algorithm, the notion of nogood is generalized so that a nogood is coupled with a *condition*, which indicates the condition in which the nogood is in effect. A condition of a nogood is a conjunction of boolean predicates each of which consists of an objective function and its cost value. For example, in the problem of Figure 1, if $x_1 = 2$ and the cost value of F_2 must be within 2, there is no possible value for x_2, since the only consistent value for x_2 is 4, where F_2 is 3. We say $\{(x_1, 2)\}$ is a nogood under the condition $F_2(S) < 3$. Note that the cost value 3 in this condition is larger than the current threshold 2, and $<$ is used instead of $\leq$, which reflects the fact that if $F_2(S) = 3$, $\{(x_1, 2)\}$ might be a part of a solution. In the modified asynchronous backtracking algorithm, agents exchange nogoods and their condition each other. When an empty nogood is found, the new lower-bound can be found by the condition of the empty nogood, i.e., if the condition is $(F_1(S) < c_1) \wedge (F_2(S) < c_2) \wedge \ldots$, the new lower-bound is the $\min_{i=1,2,\ldots} c_i$.

The DID algorithm is described as follows.

1. Perform the asynchronous backtracking.
2. If there is no solution, set c to the lower-bound obtained by the asynchronous backtracking, goto 1.
3. If a solution S is found, return S and terminate the algorithm.

For example, in the problem of Figure 1, agents first set their thresholds to 0, and try to find a solution where the costs are no greater than this threshold. By using the asynchronous backtracking algorithm, agents eventually find empty nogoods with conditions such as $(F_1(S) < 1) \wedge (F_4(S) < 1)$, and obtain the new lower bound 1. Agents then set their thresholds to 1, and try again to find a solution. Again, the asynchronous backtracking algorithm returns the new lower bound 2. After agents set their thresholds to 2, they find the solution $\{(x_1, 3), (x_2, 1), (x_3, 4), (x_4, 2)\}$, which is a worst case optimal solution.

4.3 Combination algorithm

The DDBB and DID algorithm can be combined as follows. Suppose agents are given initial lower-bound and upper-bound value, and each agent has a threshold value c and treats $F_i(S) \leq c$ as a boolean constraint.

1. Set threshold c of all agents to some value between the upper-bound and lower-bound (e.g., (upper-bound + lower-bound)/2) and perform the asynchronous backtracking.
2. If there is a solution, store the solution and set the upper-bound to the maximal cost of the solution
3. If there is no solution, set the lower-bound to the new lower-bound obtained by the asynchronous backtracking.
4. If upper-bound=lower-bound, then return the most recently stored solution, terminate the algorithm, otherwise goto 1.

4.4 Comparison of algorithms for DCOP

We use the following computation model for comparing the efficiency of DDBB, DID and the combination algorithm. We assume one time unit is required for one consistency check for each agent, and the message issued time t is available at time $t + n$, i.e., message delay is n time units.

Figure 2 shows the required time units for lazy 10 queens, varying the message delay. The initial position of the *i-th* queen is 1 for $i = 1, 3, 5, \ldots$, and 10 for $i = 2, 4, 6, \ldots$. We compare DID, DDBB, and the combination algorithm, and for each algorithm, we also show the result of the centralized version, where only one agent exists and solves the whole problem alone. For each algorithm, the distributed version outperforms the centralized version as long as the message delay is small. This is because agents can act concurrently in the distributed version of these algorithms. The amount of speedup depends on the granularity and interrelation of subproblems, as is the case in asynchronous backtracking [Yokoo, *et al.* 90]. The required time units are in the order of DID $<$ combination $<$ DDBB. This is because in this problem, the maximal cost of the worst-case optimal solution is 4, and it is closer to the lower-bound 0, rather than the upper-bound 9.

Figure 3 shows the required time units for lazy 10 queens with different initial positions. The initial position of the *i-th* queen is i. In this case, the required time units are in the order of DDBB $<$ combination $<$ DID. This is because in this problem, the maximal cost value of the worst-case optimal solution is 5, and it is closer to the upper-bound.

The tradeoffs between DDBB and DID are similar to the tradeoffs between the depth-first branch & bound algorithms and the best-first branch & bound algorithms. One desirable property of DDBB is that premature termination of the algorithm usually results a sub-optimal solution, while DID finds no

solution before the algorithm is completely finished. On the other hand, DID is an optimal algorithm in terms of the reduction of the search space under the given heuristic functions.

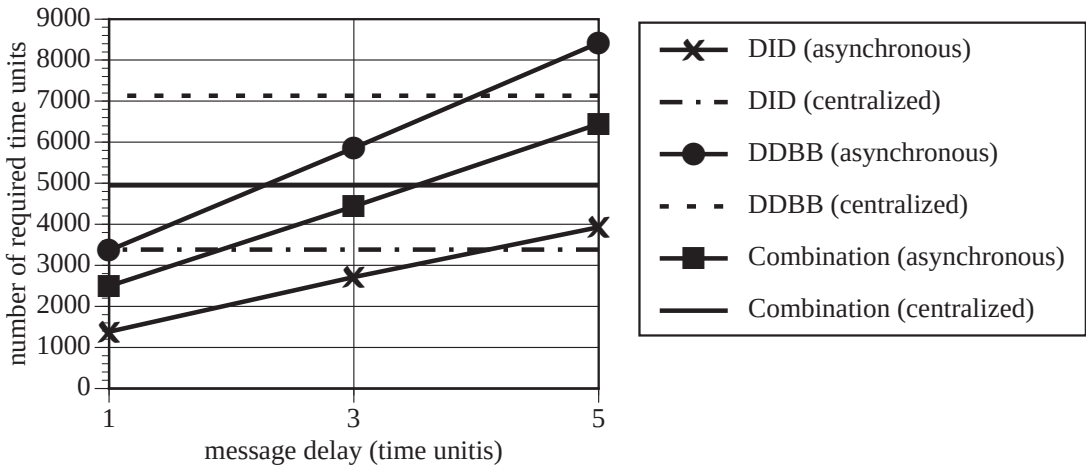

Figure 2: The Result of Lazy 10queens Problem (up-down)

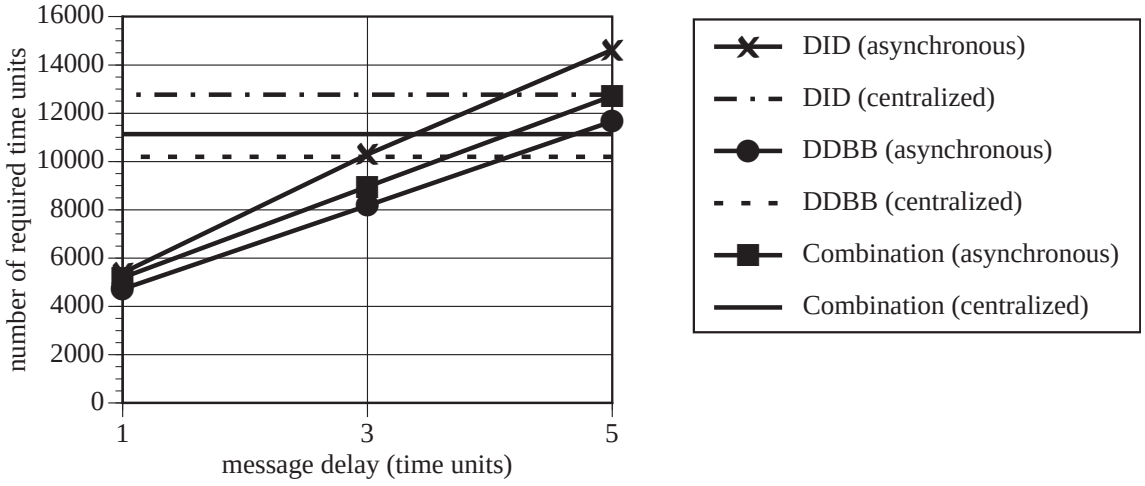

Figure 3: The Result of Lazy 10queens Problem (diagonal)

4.5 Algorithms for weighted sum optimal and pareto optimal solution

In order to obtain a weighted sum optimal solution, agents must exchange the values of the objective functions and calculate the weighted sum of the values. Suppose one agent is assigned to calculate the weighted sum, and other agents send the values of their objective functions to the agent. Then the problem is equivalent to minimizing the combined objective function. Therefore, we can use the algorithms for finding a worst case optimal solution, where only one agent has an objective function (while it requires communication between agents to calculate the value of the objective function).

We can use the variation of the DDBB algorithm for finding a pareto optimal solution. Suppose a solution S_{init} is found, then each agent i sets its threshold value to $F_i(S_{init})$, and treats $F_i(S) \leq S_{init}$ as a boolean constraint. Each agent also treats S_{init} as a nogood. If there is no solution for the corresponding DCSP, S_{init} is a pareto optimal solution. If there is a solution S_{new}, then each agent i sets its threshold value to $F_i(S_{new})$, and so on.

5 Conclusions and Future Works

In this paper, DCOP is defined as a general framework for formalizing partially adversarial/partially cooperative problems. We have discussed several solution criteria for DCOPs, and have identified relationships between this work and studies of agent rationality. We have developed the algorithms for DCOP in which agents exchange enough information to converge on a solution without exchanging all information.

Our future work will introduce time constraints (deadlines) into negotiation process [Kraus and Wilkenfeld 90], and examine the way to prevent agents from lying [Zlotkin and Rosenschein 90].

References

[Benda 86] Benda, V., Jaganathan, V. and Dodhiawala, R. : On Optimal Cooperation of Knowledge Sources, Technical Report, Boeing AT Center, Seattle, WA. 1986.

[Bond and Gasser 88] Bond, A.H. and Gasser, L. : An Analysis of Problems and Research in DAI, *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann, pp. 3–35,1988.

[Durfee and Montgomery 90] Durfee, E.H. and Montgomery, T. A. : A Hierarchical Protocol for Coordinating Multiagent Behaviors, *Proc. of AAAI-90*, pp.86–93, 1990.

[Korf 85] Korf, R.E. : Depth-first Iterative Deepening: An Optimal Admissible Tree Search, *Artificial Intelligence*, vol. 27, No.1, pp.97–109, 1985

[Kraus and Wilkenfeld 90] Kraus, S. and Wilkenfeld, J: The Function of Time in Cooperative Negotiations: In M. Huhns editor, *Proc. of 10th Workshop on Distributed Artificial Intelligence*, Chapter 4, 1990.

[Kumar 87] Kumar, V. : Branch-and-Bound Search, In S.C.Shapiro, editor, *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, pp.1003–1004, 1987.

[Lesser and Corkill 83] Lesser, V.R. and Corkill, D.D. : Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks, *AI Magazine*, pp.15–33, Fall, 1983.

[Rosenschein and Genesereth 85] Rosenschein, J.S. and Genesereth, M.R. : Deals Among Rational Agents, *Proc. of IJCAI-85*, pp.91–99, 1985.

[Sycara 85] Sycara, K.P. : Arguments of Persuasion in Labor Mediation, *Proc. of IJCAI-85*, 1985.

[Sycara, *et al.* 90] Sycara, K.P., Roth, S., Sadeh, N., and Fox, M. : An Investigation into Distributed Constraint-directed Factory Scheduling, *Proc. of CAIA-90*, 1990.

[Yokoo, *et al.* 90] Yokoo, M., Ishida, T. and Kuwabara, K. : Distributed Constraint Satisfaction for DAI Problems, In M. Huhns editor, *Proc. of 10th Workshop on Distributed Artificial Intelligence*, Chapter 18, 1990.

[Zlotkin and Rosenschein 90] Zlotkin, G. and Rosenschine, J.S., : Blocks, Lies, and Postal Freight: The Nature of Deception in Negotiation, In M. Huhns editor, *Proc. of 10th Workshop on Distributed Artificial Intelligence*, Chapter 9, 1990.