On Modeling Argumentation as Distributed Constraint Satisfaction: Initial Results

Hyuckchul Jung, Milind Tambe, Weixiong Zhang, Wei-min Shen
University of Southern California/Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{jungh,tambe,zhang,shen}@isi.edu

ABSTRACT

Conflict resolution is a critical problem in distributed and collaborative multi-agent systems. Argumentation-based negotiation, where agents provide explicit arguments or justifications for their proposals for resolving conflicts, is an effective approach to resolve conflicts. However, while small-scale argumentation systems have been developed, a well-understood computational model of argumentation is missing. The lack of such a computational model makes it difficult to investigate properties of argumentation, such as convergence and scalability, and to understand and characterize different collaborative negotiation strategies in a principled manner. To alleviate these difficulties, we adopt distributed constraint satisfaction problem (DCSP) as a computational model for investigating conflict resolution via argumentation. We model argumentation as constraint propagation in DCSP. We study convergence properties of argumentation, and formulate and experimentally compare 16 different negotiation strategies with different levels of agent cooperativeness towards others. One surprising result from our experiments is that maximizing cooperativeness is not necessarily the best strategy even in a completely cooperative environment. The results appear to have bearing not only on multi-agent argumentation, but also on general DCSP systems as well. [1]

1. INTRODUCTION

Distributed, collaborative agents are promising to play an important role in large-scale multi-agent applications including virtual environments for training, distributed robots for exploration and distributed resource scheduling. While there has been considerable research in such collaborative agents in general[4, 10], the area of collaborative conflict resolution has only recently begun to be explored. Collaborative agents may enter into conflicts over their shared resources, joint plans, or task assignments, etc, requiring effective collaborative conflict resolution techniques, particularly in large-scale applications.

Argumentation-based negotiation is a promising approach to collaborative conflict resolution[6]. In this approach, agents negotiate by providing arguments (explicit justifications) in support of their proposals to one another. Argumentation appears particularly appropriate in collaborative settings, since agents need not hide information from each other. Indeed, revealing this information is hypothesized to speed up the rate and likelihood of converging to a solution[5]. We have recently built on this previous work and developed a system called CONSA: COllaborative Negotiation System based on Argumentation[11]. CONSA is being applied in two realistic domains: battlefield simulations and distributed sensor systems. These applications require CONSA to scale-up to large numbers of agents.

While implemented argumentation systems such as CONSA have performed well in small-size applications, no systematic investigation on large-scale argumentation systems has been done. Thus, several major issues regarding the computational performance of argumentation remain unaddressed. One key open issue is understanding what impact argumentation has on conflict resolution convergence, particularly in the face of scale-up. Indeed, the presence of explicit justifications in argumentation could fail to improve convergence and may degrade performance due to processing overheads. Another key open issue is understanding different collaborative argumentation-based negotiation strategies and their impact on agent performance. In particular, what are the different principled methods in which an agent may respond to others' proposals? The answer to this question is particularly important in collaborative contexts, since well-formulated strategies from non-collaborative settings, such as threats, appeals to self interest, or attempts to undercut one's opponent[6], are inapplicable.

To address the above issues directly by building complex, large-scale agent argumentation systems is difficult and not cost-effective (e.g., what if argumentation fails after all the effort?). Furthermore, such complex systems often make it difficult to identify the critical factors that contributed to their success or failure. What is required instead is an abstract, well-understood *computational model of argumentation*, suitable for experimental investigations. To the best of our knowledge, no such computational model for argumentation has been proposed or studied. The existing ones, such as the modal logic formulations of argumentation[6], are useful tools for studying logical properties of argumentation, but not for detailed experimental investigations. Indeed, theorem proving in these logical formulations may be highly intractable and often researchers investigating such logical formulations themselves recommend more practical

[1] **This paper appeared in the Proceedings of The International Workshop on Distributed Constraint Satisfaction held in conjunction with CP 2000, September 2000**

methods of implementations.

To alleviate the above difficulties, this paper proposes distributed constraint satisfaction problem (DCSP)[13, 1] as a computational model of argumentation-based negotiation. Argumentation is modeled in DCSP as follows: when an agent communicates to others its assignments of its local variables, it also includes the local constraints that led to the assignments, as a justification. These communicated local constraints are exploited in service of constraint propagation by other agents to attempt to speed up a conflict resolution process. We focus specifically on one of the best published DCSP algorithms, that of Yokoo and Hirayama [13], and model argumentation as an extension to this algorithm by communicating local constraints. Argumentation essentially enables this DCSP algorithm to interleave constraint propagation in its normal execution.

Using this extended DCSP as our computational model, we then formulate different argumentation-based negotiation strategies, varying in the level of cooperativeness towards others. While the question of cooperativeness was raised in our implemented argumentation systems, the DCSP model enables their formalization and systematic experimentation. We specifically formulate different negotiation strategies as varying the value ordering heuristics[3]. The basic idea is to make some variable values more important than the others, so as to adjust the level of cooperativeness of an agent towards the others.

We systematically investigated 16 different negotiation strategies, conducting detailed experiments with our DCSP model. This experimentation provides the following results. First, argumentation can indeed significantly improve agents' conflict resolution, i.e., agents can more quickly resolve their conflicts and the overheads of argumentation are outweighed by its benefits (at least when the right negotiation strategy is used). Second, with respect to negotiation strategies, given that our system operates in a highly collaborative environment, the expectation was that more cooperativeness will lead to improved performance. However, a surprising result we obtain is that a maximally cooperative strategy is not guaranteed to be the most dominant strategy. Essentially, while some improvements in cooperativeness significantly improve performance, further improvements do not help and may end up degrading performance. This degradation is not only in terms of overheads but more fundamentally in negotiation cycles required to converge to a solution.

The main contribution of this paper is to employ distributed CSP technique to model multi-agent conflict resolution and more importantly to model argumentation based negotiation in conflict resolution. DCSP provides a formal tool to investigate the impact of argumentation and different argumentation-based negotiation strategies in the large-scale. Our experimental results reveal the utility of such modeling and begin to provide useful guidance for designers of argumentation systems. Additionally, the paper shows that in DCSP, argumentation inspired strategies may potentially improve performance.

2. BACKGROUND AND OPEN ISSUES

In this section, we briefly describe negotiation via argumentation based on our ongoing research of building CONSA[11]. The goals of this discussion are to present conflicts in distributed environment, explain how argumentation can be used to resolve them, and also discuss some open issues in argumentation.

2.1. NEGOTIATION WITH ARGUMENTATION

CONSA's argumentation involves an agent (sender) making a proposal to the agent-team (receivers) with an attached justification (argument). The receivers evaluate the proposal by taking the justification into account, and either accept or refute it. If refuting the proposal, a receiver may send back a counter-proposal to the team, who may continue this cycle of proposals and counter-proposals. Following [5], *negotiation objects* in CONSA refers to issues over which negotiation takes place. Agents propose and counter-propose values for these negotiation objects, with explicit justifications.

We have been working on two different domains in developing CONSA. The first is a simulation environment where agents control different distributed sensors, such as range sensors and sonars. Multiple targets appear in the environment at different intervals. The agents are required to work together to track targets under tight deadlines. As tracking tasks dynamically arrive and agents make different commitments to different tasks over time, such as what to track at a particular time interval, conflicts arise over the sensors' limited resources, including their availability, energy requirement and computational power. Such conflicts need to be resolved appropriately in order to fully utilize the sensors and optimize system performance.

The second application domain is helicopter combat simulation domain[10]. Different conflict situations arise in a team of simulated pilot agents. One example, henceforth called the *firing positions example*, involves allocating *firing positions* for a team of pilots. Individual pilots in a helicopter team typically attack the enemy from firing positions. Each firing position must be at least one kilometer apart from others, must enable a pilot to shoot at different enemy locations, and must protect the pilot from return enemy fire and yet minimize the helicopter movements. This means that a pilot's firing position is constrained by the firing positions of the others. Two firing positions are in conflict if they interfere with each other. The dynamics of the environment makes this conflict resolution complex and difficult. Each pilot may have different information of enemy, friendly vehicles and own state; and the pilot's knowledge of its environment changes dynamically. Therefore, each agent has to negotiate its position with others, rather than relying on a centralized pre-planner. A similar conflict resolution problem occurs when agents must allocate targets to attack.

To make our discussion more concrete, we now use the firing position example to describe how argumentation can be used to resolve conflicts. Consider two pilot agents, A1 and A2, and two enemy positions E1 and E2, where A1 knows about E1, while A2 knows about E2. Suppose that A1 and

A2 are only 100 meters apart, which are in conflict since they are not one kilometer apart as required. Here, agents' firing positions are the negotiation objects. A1 computes new values for the negotiation objects (positions for both agents) so as to minimize effort for both agents. It then communicates its proposal to A2, suggesting $\{$(*A1 move 450 m left, A2 move 450 m right*)$\}$, where the appended justification includes $\{$(*enemy E1 position, current separation 100 m,...*)$\}$. When A2 receives and evaluates the proposal, it realizes that it cannot move 450 meters right because of E2. A2 therefore rejects A1's proposal. It computes new positions for A1 and A2, based on the enemy position E1 sent by A1 and E2. Since the maximum A2 can move is 300 meters, it counter-proposes $\{$(*A1 move 600 m left, A2 move 300 m right*)$\}$, with the justification being that $\{$(*enemy E1 position, enemy E2 position,...*)$\}$. A1 may accept the proposal, terminating the argumentation, or it may continue argumentation.

2.2. SOME OPEN ISSUES IN ARGUMENTATION

One key shortcoming of CONSA and agent argumentation systems in general is the lack of a well-understood computational model suitable for understanding properties, such as the impact of argumentation on convergence and the overhead of argumentation. Similarly, it is difficult to formalize and investigate the impact of different negotiation strategies. For instance, in the firing position example above, pilot agents attempt to be maximally cooperative towards others, by offering to move the maximal distance they are allowed. As we scale up the number of agents, it is unclear if maximal cooperativeness will necessarily lead to improved performance.

Unfortunately, it is difficult to analyze scale-up in CONSA's current implementations directly. The state information in an agent contains large numbers of features, e.g., pilot agents have beliefs about their overall mission, organization hierarchy, standard operating procedures etc[10]. Such a large number of features makes it difficult to identify the key features and understand how they impact performance.

3. FORMALIZING ARGUMENTATION VIA DCSP

To advance the current research, we need to abstract and model the useful state characteristics relevant to argumentation. We use Distributed Constraint Satisfaction Problem (DCSP)[13, 1] as a computational model to investigate argumentation-based negotiation. DCSP allows us to easily model conflicts and conflict resolution via constraints. As a well-investigated problem, it provides efficient algorithms for conflict resolution. Most importantly, it also allows us to model the use of argumentation in conflict resolution.

A Constraint Satisfaction Problem (CSP) is commonly defined as assigning values to a list of variables V from a respective list of domains D such that a set of constraints C over the variables is satisfied. A distributed CSP is a CSP in which there are multiple agents A_i, each with a list of variables Vi, domains Di, and constraints Ci. (We consider DCSPs with multiple local variables per agent[13]). For example, consider a DCSP with two agents A_1 and A_2. Here, A_1 has $\{V1 = (x_1, x_2), D1 = (\{1,2\},\{2\}), C1 = \{(x_1 \neq x_3), (x_2 = x_1)\}\}$, and Agent A_2 has $\{V2 = (x_3), D2 = (\{1,2\}), C2 = \{(x_2 \neq x_3)\}\}$. In this DCSP, x_2 is a *locally constrained variable* for Agent A_1 because it is only constrained with variables that are local to A_1. In contrast, x_1 is an *externally constrained variable* because its value is constrained at least in part by variables in A_2. Constraints in DCSP can be similarly classified as local or external. Solving such a DCSP requires that agents not only solve their local CSP, but communicate with other agents to satisfy external constraints. A solution to the above DCSP is then A_1 assigning $(x_1, x_2) = (2, 2)$ and A_2 assigning $(x_3) = (1)$. Note that DCSP is not concerned with speeding up a centralized CSP via problem decomposition and parallelization[13]; rather, it assumes that the problem is originally distributed in the application being modeled. This assumption suits us well, since our negotiation problem is indeed a distributed problem.

We map argumentation on to DCSP as follows: we model an agents' negotiation objects as externally constrained variables, henceforth referred to as *negotiation variables*. There are external constraints among negotiation variables of different agents. All other facts and constraints that are local to an agent are modeled as the agent's local variables and constraints — these are not known by other agents. Figure 1(a) illustrates this mapping. The big circles represent agents, the squares labeled v1, v2, v3, and v4 are negotiation variables, and the small circles and the links between them are local variables and constraints. In our firing position example, the helicopter's firing positions are the negotiation variables, which constitute the current conflict and are known to the other agents. The enemy's positions are local variables because they are observed by individual agents and are not known by others. These local variables contribute to the negotiation process because they constrain the firing positions.

For our initial experimental investigations in DCSP, we abstract the mapping a bit further. We assume that each agent has only one negotiation variable, and we represent all the local variables and local constraint as a single node constraint on this negotiation variable. Illustrated in Figure 1(b), each agent has only one negotiation variable X_i and one local constraint LC_i. There is, however, no limitation on the number of external constraints C_j an agent can have. The extension of this mapping to allow multiple negotiation variables per agent is straightforward, and will be considered in our future work.

In this *abstract* mapping, an argument can be formalized in the context of DCSP as a *constraint propagation* between agents. That is, in DCSP algorithms such as Asynchronous Weak Commitment search(AWC) [13], the communication between agents is focused on the values assigned to their externally constrained negotiation variables. However, with argumentation, agents also communicate their argument (justification) in the form of local constraints (e.g., the LC_i in

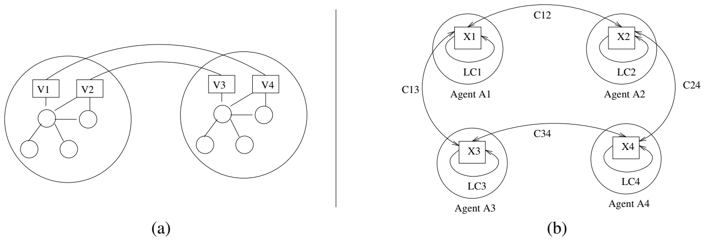


Figure 1: Model of agents in argumentation

Figure 1(b)) under which they made the selections of values for their variables. These local constraints are propagated by the agents receiving the argument. Such constraint propagation requires that when negotiation variables x_i and x_j (belonging to agents A_i and A_j respectively) share an external constraint $C_{ij}(x_i, x_j)$ then these agents know the domains of the negotiation variables. In particular, A_i is aware of x_j's domain, and A_j is aware of x_i's domain; otherwise the communication of local constraints such as LC_i may not be interpretable by others. This assumption models the situations where our pilot agents are aware of the overall region or set of positions that neighboring pilots could take, but they do not know the local constraints that restrict those positions. (Alternatively, the local constraint LC_i may explicitly outline the set of allowed values for the given negotiation variable. There are tradeoffs in the different techniques, and these may be optimized based on the domain.)

Concretely, argumentation in DCSP works as follows. Suppose an agent A_i selects a value v_i for its negotiation variable x_i. It will then send its selection v_i and its local constraint LC_i to its neighboring agent A_j with the negotiation variable x_j. Here we assume agents A_i and A_j are connected by the external constraint $C_{ij}(x_i, x_j)$ where x_i has domain D_i and x_j has domain D_j. After receiving information from A_i, agent A_j will propagate the received constraint to reduce x_j's domain D_j. This may be accomplished as follows. A_j first reduces D_i to D_i' by applying LC_i to D_i (alternatively, LC_i directly provides the values of D_i'), and A_j then it uses D_i' to reduce the size of D_j to D_j' by applying the external constraint $C_{ij}(x_i, x_j)$.

While this constraint propagation amounts only to arc-consistency, it is not run by itself to solve the DCSP, rather this is interleaved with value selection. For instance, during each cycle of AWC, we first propagate communicated constraints and then select values for variables. Thus, we also do not increase the number of communicated messages, an important issue for DCSP.

4. NEGOTIATION STRATEGIES

Given the mapping of argumentation to DCSP presented in the previous section, different negotiation strategies are considered and formalized into DCSP. A negotiation strategy refers to the decision function used by an agent to make a proposal or counter-proposal, which is a value assignment to a negotiation variable. In the mapping to DCSP, a negotiation strategy is modeled as a value ordering used to choose a value of an assignment. A value ordering heuristics rank the value of variables[3]. Different value ordering heuristics lead to different negotiation strategies.

In AWC[13], min-conflict heuristic is used for value ordering for the *good* and the *nogood* cases: min-conflict heuristic selects a variable that is in conflict, and assigns it a value that minimizes the number of conflicts (ties are broken randomly)[9]. This min-conflict heuristic is used as a baseline negotiation strategy and labeled as S_{basic}. Given the mapping of argumentation into DCSP, the S_{basic} strategy doesn't exploit argumentation in generating a more cooperative response to other agents. Argumentation enables agents to consider the constraints that the neighboring agents have on their domains, which are communicated as arguments. Taking the domains of neighboring agents into account enables an agent to generate a more cooperative response, i.e., select a value which potentially may lead to faster negotiation convergence. To elaborate on this point, we first define our notion of cooperativeness:

- **Definition**: *Cooperativeness* is defined as how much *flexibility* (or *choice of values*) is given to neighboring agents in value selection. A *more cooperative* response implies giving more flexibility to neighboring agents. More formally, let A_i be an agent with a negotiation variable X_i whose domain is D_i. If N_i is a set of agents that are A_i's neighbors and an agent A_j in N_i has n_j values from its domain (X_j) that are consistent with A_i's value v, a cooperative function f_{co} for v is defined as $f_{co}(v) = \Sigma n_j$ such that $A_j \in N_i$. It can be said that v_1 is a more cooperative response than v_2 if $f_{co}(v_1) \geq f_{co}(v_2)$. A maximally cooperative response is to select a value v_{max} such that for any other value v_{other}, $f_{co}(v_{max}) \geq f_{co}(v_{other})$.

Here, the concept of cooperativeness goes beyond merely satisfying their constraints and enables even faster convergence. That is, an agent A_i can provide a more cooperative response to a neighbor agent A_j, by selecting a value for its negotiation variable that not only satisfies the constraint with A_j, but maximizes flexibility for A_j. If A_i is facing a nogood, and gives A_j more choice by selecting v_{max}, then A_j can more easily select a value that satisfies A_i and A_j's other external constraints (for instance, if there is another agent A_k constraining A_j). Giving more choice to A_j in *good* case also appears useful, since A_j may then more flexibly negotiate with others, without violating the external constraint with A_i. This is indeed partly the rationale for the helicopter pilots such as A2 in the firing position example (described in Section 2) to offer the maximum flexibility to their teammates such as A1. Having lower possibility of constraint violation, this cooperative response can lead to faster convergence.

S_{basic} tries to minimize the number of conflicts without taking neighboring agents' own restrictions into account for value ordering. An agent A_i's selected value v with S_{basic} can have a smaller f_{co} value, compared with v_{max} in D_i: that is, $f_{co}(v) \leq f_{co}(v_{max})$. That is, in terms of flexibility, S_{basic} is not guaranteed to select a maximally cooperative response. Hence, S_{basic} is not the most cooperative strategy to neighbor agents.

Based on the cooperativeness defined above, different strategies other than S_{basic} can be introduced and formalized in terms of value ordering (as discussed below): an agent A_i can rank each value (v) in its domain D_i based on how much flexibility is given to its neighbors with v when it assigns a value for its negotiation variable X_i. These strategies rely on the basic framework from AWC.

Different negotiation strategies are described in terms of the *good* and *nogood* cases because different value ordering methods can be applied in both cases. To explain the negotiation strategies, let N_i^{high} (N_i^{low}) be the neighbor agents of A_i whose negotiation variable's priority is higher (lower) than the priority of A_i's negotiation variable X_i. In the *good* case, an agent A_i computes a set (V_i) of consistent values for its negotiation variable X_i from its domain D_i. For each value v in the set V_i, A_i computes a number n^{high} and n^{low}. n^{high} (n^{low}) is the sum of the number of consistent values with v for each agent in N_i^{high} (N_i^{low}). Having computed n^{high} and n^{low}, three different negotiation strategies according to cooperativeness can be considered for the *good* case. Each of them is described as follows:

- **S_{high}**: each agent selects a value from its domain which maximizes n^{high} to provide maximal flexibility to the neighbor agents in N_i^{high}. In this way, an agent A_i attempts to be maximally cooperative towards its higher priority neighbors.

- **S_{low}**: each agent selects a value from its domain which maximizes n^{low} to provide maximal flexibility to the lower priority agents in N_i^{low}.

- **S_{all}**: let n^{all} be the sum of n^{high} and n^{low}. To provide maximal flexibility to all neighbor agents, each agent selects a value from its domain which maximizes n^{all}.

- **S_{basic}**: the original AWC search algorithm based on min-conflict heuristic as described above.

Here, strategy S_{all} is the most cooperative strategy because it gives maximal flexibility to all of an agent(A_i)'s neighbors. S_{high} (or S_{low}) counts only the flexibility of the agents in N_i^{high} (or N_i^{low}). The maximal flexibility of S_{high} (or S_{low} or S_{basic}) is equal to or less than that of S_{all}. In particular, the function f_{co} sums up all the flexibility of all neighboring agents for a given value and S_{all} selects a value which maximizes n^{all}. Therefore, based on the definition of cooperativeness, S_{all} is more cooperative than S_{high} (or S_{low} or S_{basic}). Both S_{high} and S_{low} have trade-offs. For instance, S_{high} may leave very little or no choice to an agent's neighbors in N_i^{low}, making it impossible for them to select any value for their negotiation variables. S_{low} has a converse effect. S_{basic} also has trade-offs because it does not pay attention to flexibility of neighboring agents.

The computation in the *nogood* case is identical to the *good* case except that the set V_i is the set of all values in D_i, not the set of consistent values. Note that, in this *nogood* case, X_i's priority is increased as usual, and n^{high} and n^{low} are based on the variable's priority prior to this increase. A question here is "why should we consider neighboring agents' prior priority?". Answer is: highly constrained neighboring agents tend to have higher priorities than less constrained agents. If an agent is more cooperative to the highly constrained agents, they will have less chances to turn into *nogood* in the next cycle, which can lead to fast convergence to a solution. Because current priorities reflect the degree of agents' being constrained, the priority prior to the increase is used to group neighboring agents into higher and lower priority groups. Thus, three different strategies above can be also considered in the *nogood* case.

Based on the ideas introduced above, we can generate different strategy combinations by choosing and combining different negotiation strategies for the *good* and the *nogood* cases: there are 16 possible combinations from 4 different negotiation strategies(S_{high}, S_{low}, S_{all}, and S_{basic}) for each *good* case and *nogood* case. In the experiments, all of the possible 16 combinations are systematically examined for completeness. In the following, some examples of negotiation strategy combinations used for experiments are described. Each combination below is described in terms of its response in the *good* and *nogood* cases. *Note that all the strategies are enhanced with argumentation (constraint propagation).*

- **S_{basic}-S_{basic}**: This is the original AWC. Min-conflict heuristic is used for the *good* and *nogood* case.

- **S_{low}-S_{high}**: For the *good* case, an agent is maximally cooperative towards its lower priority neighbor agents by using S_{low}. While giving maximal flexibility to lower neighbors, the selected value doesn't violate the constraints with higher neighbors. On the contrary, for the *nogood* situations, an agent attempts to be maximally cooperative towards its higher priority neighbors by using S_{high}.

- **S_{all}-S_{all}**: In both the *good* and the *nogood* cases, an agent uses S_{all} for the value ordering, which is to select a value that attempts to maximize flexibility of all neighbor agents.

The above are only three examples out of 16 strategy combinations that we experiment with in the next section. Because S_{all} is more cooperative than S_{low} (or S_{high}), it can be said that S_{all}-S_{all} is the most cooperative strategy combination. Figure 2 shows a partial order over the cooperativeness of 16 different strategy combinations. The higher combinations are more cooperative than the lower ones. The combinations at the same level are not comparable to each other such as S_{low}-S_{high} and S_{high}-S_{low}. While S_{basic}-S_{basic} is not comparable to other strategy combinations such as S_{low}-S_{high} in the same level, this S_{basic}-S_{basic} was not originally defined with the notion of cooperativeness as defined in this section; while a strategy combination such

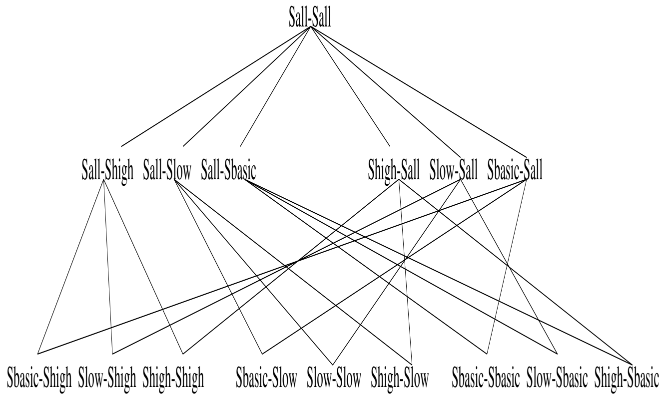


Figure 2: Cooperativeness relationship

as S_{low}-S_{high} attempts to be explicitly cooperative to neighboring agents in the sense of the definition of cooperativeness discussed earlier.

5. EXPERIMENTAL EVALUATION

In this section, we experimentally study the benefits of using DCSP as a model of conflict-resolution using argumentation. We investigated the impact of argumentation in speeding up conflict resolution processes and the computational performance of the negotiation strategies from the previous section.

DCSP experiments in this work were motivated by the firing position example. In the experiments, each pilot was modeled as an agent; and each agent had one negotiation variable to model the pilot's firing position. The domain of this variable was the set of positions the pilot could take. However, the domain was restricted by local node constraints, which were the constraints imposed by the enemy positions visible to the pilot. The negotiation variable also had external constraints with the negotiation variables of its neighboring agents (where the neighbors were determined as discussed below). This external constraint modeled the real-world constraint that, if two pilots were neighbors, then their positions must at least be some fixed distance from each other. If this external constraint was violated, then clearly, the pilots' positions were in conflict with each other. Based on these mapping, a distributed constraint satisfaction problem for firing position example was constructed, and the goal of the argumentation-based negotiation was to find values for agents' negotiation variables that satisfied all of their local and external constraints.

Four different types of DCSP configurations were considered in the experiments: a chain, a ring, a tree and a grid. In the chain configurations, each agent had two neighboring agents, to its right and left (except for end points). Since there was a constraint between the negotiation variables of neighboring agents, the negotiation variables essentially formed a chain. The ring configuration added a constraint between the negotiation variables of the first and the last agents of the chain configuration. Fig. 1b shows a very small ring configuration, with four agents. In the tree configuration, agents' negotiation variables fit into the nodes of a binary tree and had constraints with their parents and children. Finally, in a grid configuration, the negotiation variables formed a grid in which a variable was constrained by its four neighbors except the ones on the grid boundary.

Given a DCSP, a centralized approach can be the most trivial method to solve the problem because there has been considerable research in CSP and many CSP algorithms are available. All agents could communicate all their local information to a single agent which could solve all the conflicts using a constraint satisfaction algorithm. However, in many applications, such a centralized approach could prove problematic for a variety of reasons. First, this approach introduces a central point of failure, so that there is no fault tolerance. If the centralized agent somehow fails (failures are likely in a real-world environment), the entire system will come to a halt. Second, centralization of all information could be a significant security risk, open to actual physical or cyber-attacks, particularly in hostile adversarial environments. Third, a centralized agent could be significant computational and communication bottleneck. Specifically, in domains such as distributed sensors, negotiations must continuously occur among all agents for continual readjustment of the sensors. Centralization would require all sensors to repeatedly and continuously communicate their local information to the centralized agent for the centralized computation. With hundreds or even thousands of agents communicating such information, there is a significant potential for the centralized agent to be a computational and communication bottleneck. A distributed system provides fault tolerance, reduces the security risk and avoids a centralized communication/computational bottleneck. So, we believe the centralized approach is not suitable for multi-agent systems, especially for our application domains of distributed sensors and helicopter pilot simulation.

Our experiments followed the method used in [13] and same criteria were used for evaluation. The experimental results reported below were averaged over 100 runs and all the problem instances were solvable. The number of agents was 512 and the negotiation variable of each agent had one dozen values in its domain.

5.1. PERFORMANCE OF NEGOTIATION STRATEGIES

For the performance evaluation of the strategies described in Section 4, those strategies were compared on each of the four DCSP configurations (chain, ring, tree and grid) described in the above. The main goal was to investigate the impact of the different cooperative strategies on negotiation performance.

The main criterion for the evaluation was the time cost because running time was very critical to our applications such as battlefield simulations. Regarding the time cost, *cycles* and *constraint checks* were measured as in [13]: *cycles* is the number of cycles consumed until a solution is found, and *constraint checks* is the sum of the maximal numbers of constraint checks performed by agents at each of the negotiation cycle. Figure 4 shows the results for *constraint checks*,

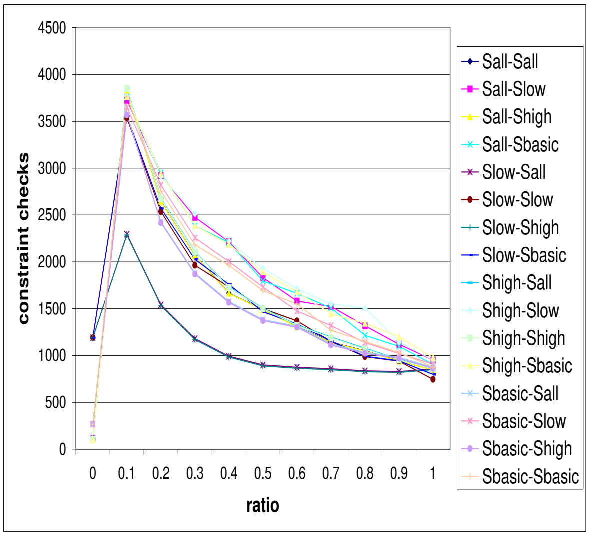


Figure 3: Comparison of negotiation strategies: constraint checks

where the figures from (a) to (d) are the results for a chain, a ring, a tree and a grid, respectively. The horizontal axes are the ratios of the number of locally constrained agents to the total number of agents. Each locally constrained agent has a local constraint (described in Section 3) which randomly restricts available values for its negotiation variable. Thus, for example, local constraint ratio 0.1 means that 10 percent of the agents have local constraints. Having local constraints, agents have less flexibility to assign a value to their negotiation variables. The vertical axes are *constraint checks*.

Experiments were performed for the sixteen negotiation strategy combinations described in Section 4. *Constraint checks* on the chain configuration is shown in Figure 3. The results show that S_{low}-S_{high} was the best, and the results also show that those strategy combinations with S_{basic} and S_{low} for *nogood* response (e.g., S_{all}-S_{low}) performed worse than the others. *Cycles* on the chain configuration showed the same pattern as *constraint checks* did.

Given 16 strategy combinations, it is difficult to understand different patterns in Figure 3. For expository purposes, we will henceforth present the results from four specific strategies. First, S_{all}-S_{all} is selected because it is the most cooperative strategy combination. Second, the original AWC strategy (S_{basic}-S_{basic}) is selected to compare it with other negotiation strategies. Third, S_{low}-S_{high} which shows the best performance is selected. Lastly, S_{high}-S_{low} is selected because it is the worst performing strategy and the opposite to the best performing strategy. Using these four strategies does not change the conclusions from our work, rather it is done solely to make it easier to present the graphs and discuss the results.

The graphs in Figure 4 show the *constraint checks* of the selected strategies on each of the four configurations (tree, ring, tree, and grid) described above. These graphs show an interesting result that maximal cooperativeness to neighbor agents was not a dominant strategy: in S_{all}-S_{all}, each agent considers all of its neighbors in both the *good* and the *nogood* cases. Though S_{all}-S_{all} performed better than S_{basic}-S_{basic}, it was worse than S_{low}-S_{high} in the chain and the ring configurations. On the contrary, S_{low}-S_{high} showed the best performance in all of the configurations except for the ratio of 0.0. Though S_{low}-S_{high} was a winning strategy in terms of *constraint checks*, there could be a possibility that S_{low}-S_{high} performed worse in terms of *cycles*. However, the results in Figure 5 eliminates such a possibility because *cycles* show the same pattern as *constraint checks* do (Figure 4). That is, S_{low}-S_{high} was the best strategy for *cycles*, too. Here, one assumption was that communication cost was mainly dependent on the number of communications rather than message size. Hence, communication of local constraints was not counted as extra cost.

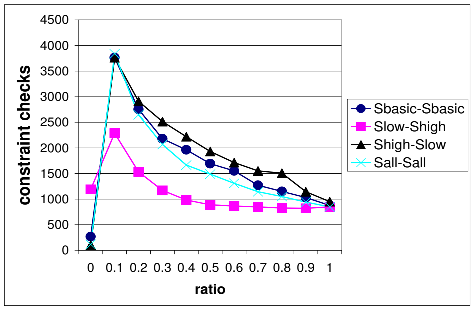
The results are surprising because we expected that the most cooperative strategy combination S_{all}-S_{all} would be more dominant. However, the less cooperative strategy combination S_{low}-S_{high} showed the best performance. So, we conclude that certain level of cooperativeness is useful, but maximal cooperativeness is not necessarily the best negotiation strategy. The results also show that S_{basic}-S_{basic} which didn't benefit from argumentation couldn't perform as well as S_{low}-S_{high} which exploited argumentation.

5.2. BENEFITS OF ARGUMENTATION

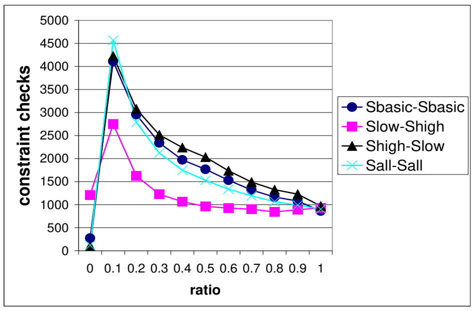
One critical question to be answered was how much total amount of conflict resolution effort was saved by incorporating argumentation in negotiation, and whether the overhead of argumentation could be justified.

To answer this question, two different versions of S_{low}-S_{high} were compared. The first version was the S_{low}-S_{high} described in Section 4. The second version was same with S_{low}-S_{high} except that it didn't use any argumentation (constraint propagation). Let this second strategy combination be S_{low}-S_{high}(*noarg*). S_{low}-S_{high} was chosen because it was the best among all the strategy combinations to be considered. In S_{low}-S_{high}, each agent received arguments (local constraints) from neighbor agents and used the propagated constraints for eliminating its local problem space. However, the agent had an overhead of checking extra constraints. In S_{low}-S_{high}(*noarg*), agents did not receive arguments from neighbors, and thus did not have to propagate constraints.

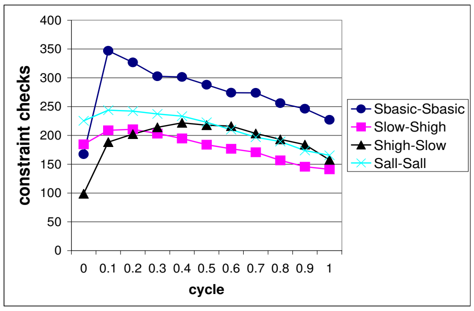
Figure 6 shows the experimental results for the chain configuration with 16 agents: with 512 agents, S_{low}-S_{high}(*noarg*) exceeded the *cycles* limit (10,000) for the most part. As in [12], agents saved the whole agent view as a nogood in both S_{low}-S_{high} and S_{low}-S_{high}(*noarg*) because finding all minimal nogoods [8] required certain amount of computation cost. The results for the other configurations were similar. Argumentation helped S_{low}-S_{high} to reduce the total negotiation effort as measured by *constraint checks* (Figure 6.a) and *cycles* (Figure 6.b).



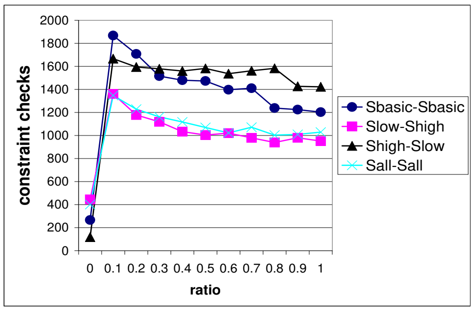
(a) Chain



(b) Ring

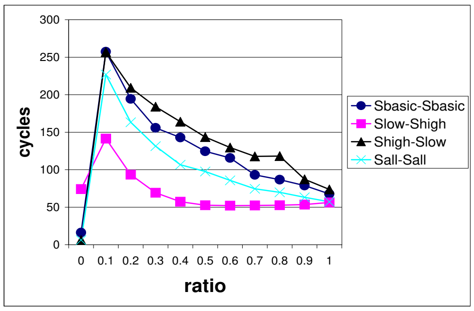


(c) Tree

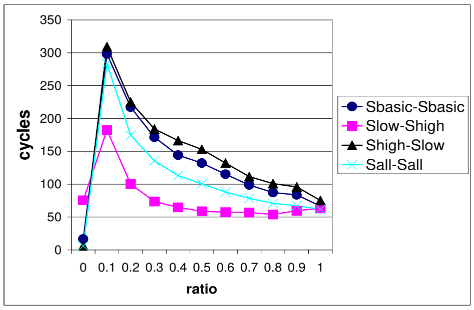


(d) Grid

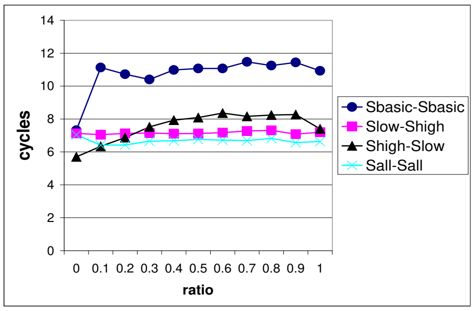
Figure 4: Comparison of negotiation strategies: constraint checks



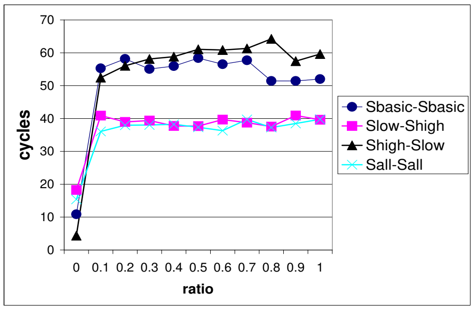
(a) Chain



(b) Ring

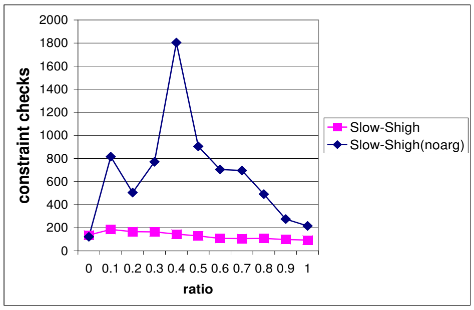


(c) Tree

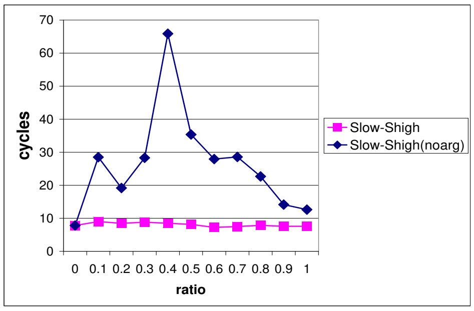


(d) Grid

Figure 5: Comparison of negotiation strategies: number of negotiation cycles.

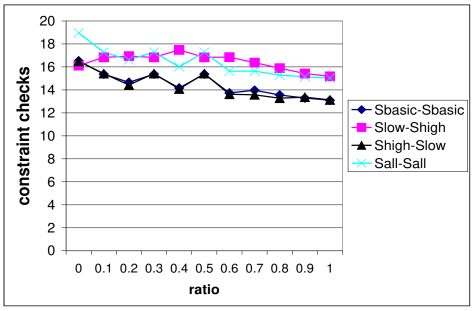


(a)

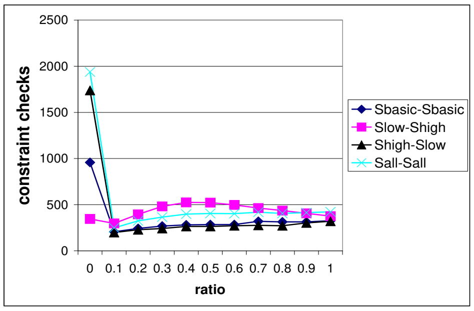


(b)

Figure 6: Benefit of argumentation: A computational comparison.



(a) maximal



(b) average

Figure 7: Maximal and average effort per cycle.

Argumentation could improve conflict resolution performance and its overheads appeared to be well justified. However, one interesting question from the results in Figure 4 and Figure 5 was how a negotiation strategy combination, especially S_{low}-S_{high}, could reduce the number of cycles and total work at the same time. Figure 7 provides an answer. It presents the maximal and average workload of these strategy combinations in all cycles. It shows that S_{low}-S_{high} gave the highest workload by cycle among all strategy combinations, which indicated that more agents were busy at each cycle. In other words, negotiation effort was distributed more evenly among agents by S_{low}-S_{high} than the others.

6. RELATED WORK

While this paper builds on several previous efforts in argumentation[6] and distributed constraint satisfaction[13], it is a unique effort in synthesizing these two areas. Argumentation has been rigorously investigated using different logics[6], including specially designed logics of argumentation. However, such formalization appears inappropriate for empirical investigation of the effects of argumentation on conflict resolution convergence, or the effects of different negotiation strategies. In contrast, we have proposed DCSP as a model of argumentation, enabling systematic experimental investigation of the computational properties of argumentation systems in the large-scale.

Our work has built on the rich foundations of the existing DCSP work[1, 12, 13]. Our ability to experimentally investigate argumentation and negotiation strategies is a testimony to the effectiveness of using DCSP as a computational model. We have modeled argumentation as constraint propagation

Since agents in our work focus in part on conflict resolution over limited resources (although they may also negotiate about joint plans, or tasks), research on distributed resource allocation is also related. While resource allocation is itself a broad area of research, our use of argumentation for resource conflict resolution and the use of DCSP for modeling such conflict resolution sets our work apart. For instances, [7] extends dispatch scheduling to improve resource allocation; and [2] on distributed scheduling airport-ground scheduling service. While these systems do not perform conflict resolution via argumentation, hopefully our investigations will begin to shed light on the utility of argumentation in these domains. Finally, whether argumentation could enhance current market-based approaches in non-collaborative settings remains an open issue for the future.

7. CONCLUSION

Argumentation is an important conflict-resolution technique in multi-agent research. Several researchers, including ourselves, have developed concrete implemented argumentation systems. However, much of the existing work has focused on smaller-scale systems, and major issues regarding the computational performance of collaborative argumentation, particularly in the presence of scale-up, remain unaddressed. Yet it is difficult to work with the complex implemented argumentation systems to directly resolve these issues.

The contributions of this paper mainly focus on multi-agent conflict resolution via argumentation. We chose to model argumentation in terms of distributed constraint satisfaction problem (DCSP), which provides a well-understood computational platform to address the issues above. The key contributions of this paper are: (1) modeling of argumentation in terms of constraint propagation in DCSP; (2) modeling of different argumentation strategies and investigating different degrees of cooperativeness using value ordering techniques of CSP. We also computationally compared collaborative negotiation strategies.

The utility of DCSP is seen in our ability to conduct detailed experiments to address the issues raised, and quantitatively measure the performance of argumentation and different negotiation strategies. These results show that argumentation indeed leads to faster convergence of conflict resolution at least given the right resolution strategy. Our experiments also revealed a surprising result that the most cooperative argumentation strategy may not be necessarily the best *in terms of convergence* in negotiation. Furthermore, applying constraint propagation and value ordering to the existing DCSP algorithms, as suggested in this paper, appears to improve these algorithms.

8. ACKNOWLEDGEMENTS

This research is funded by DARPA ITO award number F30602-99-2-0507.

REFERENCES

[1] Aaron Armstrong and Edmund H. Durfee. Dynamic prioritization of complex agents in distributed constraint satisfaction problems. In *Proc. of the Intl. Joint Conference on Artificial Intelligence*, August 1997.

[2] Mike H. Chia, Daniel E. Neiman, and Victor R. Lesser. Poaching and distraction in asynchronous agent activities. In *Proc. of the Third International Conference on Multi-Agent Systems (ICMAS)*, July 1998.

[3] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-97)*, August 1995.

[4] B. Grosz. Collaborating systems. *AI magazine*, 17(2), 1996.

[5] N.R. Jennings, S. Parsons, P. Noriega, and C. Sierra. On argumentation-based negotiation. In *Proc. of the International Workshop on Multi-Agent Systems*, 1998.

[6] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–70, 1998.

[7] Jyi-Shane Liu and Katia P. Sycara. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS)*, July 1996.

[8] Dorothy L. Mammen and Victor R. Lesser. Problem structure and subproblem sharing in multi-agent systems. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-98)*, 1998.

[9] S. Minton, M. D. Johnston, A. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the National Conference on Artificial Intelligence*, 1990.

[10] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124, 1997.

[11] M. Tambe and H. Jung. The benefits of arguing in a team. *AI Magazine*, 20(4), 1999.

[12] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.

[13] Makoto Yokoo and Katsutoshi Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proc. of the Third Intl. Conf. on Multi-Agent Systems(ICMAS)*, July 1998.