

Argumentation for Coordinating Shared Activities

Bradley J. Clement, Anthony C. Barrett, and Steven R. Schaffer

Jet Propulsion Laboratory

California Institute of Technology

4800 Oak Grove Drive, M/S 126-347

Pasadena, CA 91109-8099

{bclement, barrett, srschaff}@aig.jpl.nasa.gov

Abstract

There is an increasing need for space missions to be able to collaboratively (and competitively) develop plans both within and across missions. In addition, interacting spacecraft that interleave onboard planning and execution must reach consensus on their commitments to each other prior to execution. In domains where missions have varying degrees of interaction and different constraints on communication and computation, the missions will require different coordination protocols in order to efficiently reach consensus within their imposed deadlines. We describe a Shared Activity Coordination (SHAC) framework that provides a decentralized algorithm for negotiating the scheduling of shared activities over the lifetimes of multiple agents and a foundation for customizing protocols for negotiating planner interactions. We investigate variations of a few simple protocols based on argumentation and distributed constraint satisfaction techniques and evaluate their abilities to reach consistent solutions according to computation, time, and communication costs in an abstract domain where spacecraft propose joint measurements.

1 Introduction

When interleaving planning and execution, an agent adjusts its planned activities as it gathers information about the environment and encounters unexpected events. Interacting agents coordinate these adjustments to manage commitments with each other. Demand for this kind of autonomous agent technology is growing for space applications. Autonomous spacecraft promise new capabilities and cost improvements in exploring the solar system. Spacecraft (and rovers) that explore other planets have intermittent, delayed communication with Earth, requiring that they be able to manage their resources and operate for long periods in isolation.

In addition, there is a growing trend toward multi-spacecraft missions. These spacecraft will coordinate measurements, share data, and route data to and from Earth. Separate missions, such as those to Mars have their own budgets, experiments, and operations teams. As such, the spacecraft represent self-interested entities that benefit from collaborative interactions.

But, even a single spacecraft has multiple science instruments for executing different goals of different scientists, and different operations groups will have different areas of expertise over different

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

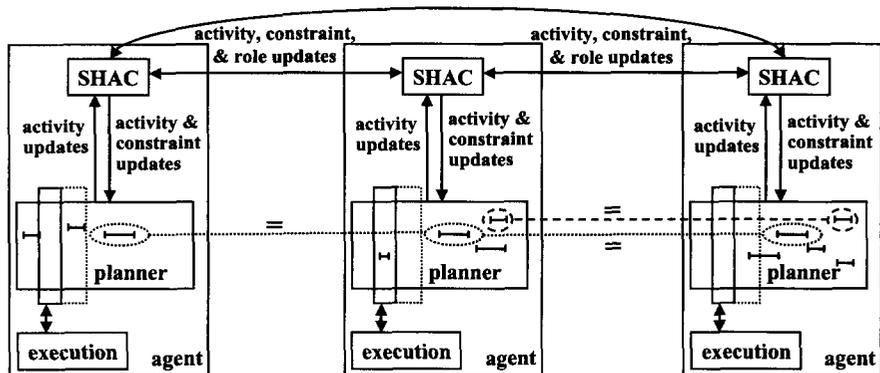


Figure 1: Shared activity coordination

subsystems for control. These different groups negotiate over mission plans in the same way that different Mars missions must collaborate over spacecraft interactions. Whether this negotiation is done on-board or on Earth, there is a distributed operations planning problem that benefits from automation. Both also have real-time aspects. On-board systems must plan safely over near- and long-term horizons, and ground systems must also replan based on changing contexts in daily, weekly, and lifelong mission exercises. Ground planning also suffers from communication constraints. Scientists from different universities or opposite sides of the globe will intermittently provide inputs and respond on an irregular basis. A collaboration/negotiation system must be built around communication constraints to meet hard deadlines for coming to consensus on consistent operations plans.

The field of multiagent planning has largely focused on fully cooperative planning and execution [5, 6, 15, 9, 3]. Market-based agent systems address near-term resource negotiation but have rarely addressed how near-term decisions affect longer-term goals. Multiagent systems built for Robocup Soccer competitions mainly address collaborative multiagent execution in an adversarial environment and have limited planning capabilities. These approaches do not adequately address real-time planning for self-interested agents.

Argumentation is a technique for negotiating joint beliefs or intentions [11] among cooperative or self-interested agents. Commonly, one agent makes a proposal to others with justifications. The others evaluate the argument and either accept it or counter-propose with added justifications. This technique has been applied to teamwork negotiation to form teams, reorganize teams, and resolve conflicts over members' beliefs [16]. It can also be used to establish consensus on shared activities.

We present a framework for Shared Activity Coordination (SHAC) based on argumentation techniques for negotiation. SHAC consists of an algorithm for continually coordinating agents and a foundation for rapidly designing and implementing coordination protocols based on a model of shared activities. The treatment of reaching consensus in real-time is discussed in [2]. We describe distributed planning mechanisms (protocols) built on ideas of argumentation and one based on distributed constraint satisfaction techniques employing argumentation [10]. We evaluate them according to computation, time, and communication costs in a distributed spacecraft domain where joint measurements are proposed, and capabilities applied are negotiated. Our ultimate goal is to create interacting agents that autonomously adjust their coordination protocols with respect to unexpected events and changes in communication or computation constraints so that the agents can most efficiently achieve their goals.

2 SHAC

Our approach, called Shared Activity Coordination (SHAC), provides a general algorithm for interleaving planning and the exchange of plan information based on shared activities. Agents coordinate their plans by establishing consensus on the parameters of shared activities. Figure 1 illustrates this approach where three agents share one activity and two share another. The constraints denote equality requirements between shared activity parameters in different agents. The left vertical box over each planner's schedule represents a *commit window* that moves along with the current time. A consensus window is shown to the right of the commit window, within which consensus must be quickly established before committing. Since consensus is hard to maintain when all agents can modify a shared activity's parameters at the same time, agents must participate in different coordination roles that specify which agent has control of the activity. As shown in the figure, SHAC interacts with the planning and execution by propagating changes to the activities, including their parameters and constraints on the values of those parameters.

2.1 Shared Activities

The model of a shared activity is meant to capture the information that agents must share, including control mechanisms for changing that information. A shared activity is a tuple (*parameters, agent roles, protocols, decomposition, constraints*). The parameters are the shared variables and current values over which agents must reach consensus by the time the activity executes. The agent roles determine the local activity of each agent corresponding to the joint action. To provide flexible coordination relationships, the role activities of the shared activity can have different conditions and effects as specified by the local planning model. The shared parameters map to local parameters in the role activity.

For example, a shared data communication activity can map to a `receive` role activity for one agent and a `send` role activity for another. Shared parameters could specify the start time, duration, transfer rate, and data size of the activity. The data size is depleted from the sender's memory resource but added to the receiver's memory. The agents could have separate power usages for transmitting and receiving.

Protocols are the mechanisms assigned to each agent (or role) that allow the agents to change constraints on the shared activity, the set of agents assigned to the activity, and their roles. Constraints will be described in the next section, and a variety of protocols will be defined in the Protocols section.

The shared decomposition enables agents to select different team methods for accomplishing a higher level shared goal. Specifically, the decomposition is a set of shared subactivities. The agents can choose the decomposition from a pre-specified set of subactivity lists. For example, a joint observation among orbiters could decompose into either (`measure, process_image, downlink`) or (`measure, downlink`).

2.2 Constraints

Constraints are created by agents' protocols to restrict sets of values for parameters (*parameter constraints*) and permissions for manipulating the parameters, changing constraints on the parameters, and scheduling shared activities (*permission constraints*). These constraints restrict the privileges (or responsibilities) of agents in making coordinated planning decisions. By communicating constraints, protocols can come to agreement on the scheduling of an activity without sharing all details of their local plans.

Given: a *plan* with multiple activities including a set of *shared_activities* with *constraints* and a *projection* of *plan* into the future.

1. Revise *projection* using the currently perceived state and any newly added goal activities.
2. Alter *plan* and *projection* while honoring *constraints*.
3. Release relevant near-term activities of *plan* to the real-time execution system.
4. For each shared activity in *shared_activities*,
 - apply each associated protocol to modify the shared activity;
5. Communicate changes in *shared_activities*.
6. Update *shared_activities* based on received communications.
7. Go to 1.

Figure 2: Shared activity coordination algorithm

A parameter constraint is a tuple $\langle agent, parameter, value\ set \rangle$. The *agent* denotes who created the constraint. Some protocols differentiate their treatment of constraints based on the agent that created them. For example, the asynchronous weak commitment algorithm prioritizes agents so that lower-priority agents only conform to higher-priority agent constraints [17]. Agents can add to their constraints on a parameter, replace constraints, or cancel them. A string parameter constraint, for example, can restrict a parameter to a specific set of strings. An integer or floating point variable constraint is a set of disjoint ranges of numbers. Scheduling constraints can be represented as constraints on a start time integer parameter.

Permission constraints determine how an agent's planner is allowed to manipulate shared activities. Permissions can be defined for adding, moving, deleting, choosing refinements, or modifying parameters of a shared activity.

2.3 Coordination Algorithm

The purpose of the SHAC algorithm is to negotiate the scheduling and parameters of shared activities until consensus is reached. Figure 2 gives a general specification of the algorithm. SHAC is implemented separate from the planner, so steps 1 through 3 are handled by the planner through an interface to SHAC. Step 4 invokes the protocols that potentially make changes to refocus coordination on resolving shared activity conflicts and improving plan utility. SHAC sends modifications of shared activities and constraints to sharing agents in step 5. In step 6, shared activities and constraints are updated based on changes received from other agents.

3 Protocols

In general, protocols determine when to communicate, what to communicate, and how to process received communication. During each iteration of the loop of the coordination algorithm (Figure 2), the protocol determines what to communicate and how to process communication. A protocol is defined by how it implements the following procedures to be called during step 4 of the SHAC coordination algorithm for the shared activity to which it is assigned:

1. modify permissions of the sharing agents
2. modify locally generated parameter constraints
3. add/delete agents sharing the activity

4. change roles of sharing agents

The default protocol, representing a base class from which other protocols inherit, does nothing for these methods. However, even with this passive protocol, the SHAC algorithm still provides several capabilities:

joint intention A shared activity by itself represents a joint intention among the sharing agents.

mutual belief Parameters or state assertions of shared activities can be updated by sharing agents to establish consensus over shared information.

resource sharing Sharing agents can have identical constraints on shared states or resources.

active/passive roles Some sharing agents can have active roles with execution primitives while others have passive roles without execution primitives.

master/slave roles A master agent can have permission to schedule/modify an activity that a slave (which has no permissions) must plan around.

For convenience, we will refer to this abstract protocol as *chaos* because it can allow multiple agents to make changes to the same shared activity concurrently. This can lead to thrashing if the agents undo each others contributions to solving the problem. This can also lead to inconsistent information. For example, if agent A sends " $x = 0$ " to B and C, B gets A's message and sends " $x = 1$ " to A and C, and C gets B's message before A's, then C will believe $x = 0$ while A and B believe $x = 1$. If unchecked, the agents may incorrectly believe consensus has been reached. This is a violation of *causal consistency* [13]. An additional requirement is that messages from any one agent are received in the order sent. This is *atomic consistency*. For our experiments, we use TCP/IP communications, which only guarantee atomic consistency.

The following sections describe some subclasses of this abstract protocol focusing on regulating control through permissions. These protocols can be further adapted in their handling of constraints and agent roles, as will be discussed in Section 4. These are the basic ingredients of argumentation-based negotiation.

3.1 Master/Slave Protocol

While SHAC allows masters and slaves to be defined by initially specified permissions, we briefly describe a protocol subclass from which others that we discuss inherit. This master/slave protocol avoids the thrashing and consistency problem of chaos by only giving one agent permission to modify the activity. It simply assigns the creator of a shared activity the master and all others slaves. Slaves sometimes need basic permissions on a shared activity in order to add them to the plan. For example, in interfacing to the ASPEN planning system, we found that a "detail" permission was needed to refine a goal into detailed activities. Master and slave permissions are initialized for each shared activity type and assigned to the agents at activity instantiation.

This protocol is appropriate for centralizing decisions at the level of a shared activity. Slave agents must trust the master and are thus cooperative. A problem with this protocol is that the master may not receive enough information about the local constraints of the other agents in order to find a solution. Without feedback from the slaves, the master can settle on a locally consistent solution and not know that its choices cause irreparable flaws with the slaves. Thus, the search space can be limited, sacrificing both completeness and optimality.

3.2 Round Robin Protocol

A round-robin approach to establishing consensus on a shared activity involves rotating a master role by changing permission constraints. This protocol gets around the search space limitation of master/slave by enabling all sharing agents to contribute to the solution. Thus, this protocol is applicable to self-interested agents. Only one agent may modify the activity at a time and once finished, the agent can turn off its own permissions and turn them on for another agent (while sending out the update). The round robin protocol can inherit from the master/slave protocol and can be implemented by the following method. *time_elapsed* can be a parameter of the shared activity that is updated by the current master agent.

Round-Robin modifyPermissions method

- if have master permissions
 - update *time_elapsed*
 - if finished planning or *time_elapsed* > threshold
 - * restrict self to slave permissions
 - * add master permissions for next agent
 - * set *time_elapsed* to 0

3.3 Asynchronous Weak Commitment

Multi-asynchronous weak commitment is an algorithm for solving distributed constraint satisfaction problems (DCSPs) that enables agents, each with a set of variables, to satisfy constraints between variables across and within agents [17]. Agents are prioritized, and their variables each initially have a zero priority. The values of lower priority variables are modified to satisfy constraints with values chosen for higher priority variables (with agent priorities as a tie breaker). If there is no value that satisfies the constraints, then the governing agent selects a value that minimizes violations with lower priority variables and raises the priority of the variable to one higher than the highest priority of the variables with which it has constraints, making the variable the highest ranking with its neighbors. The failing agent also sends a *no-good* to its neighbors, communicating the values of the subset of variables making the variable unassignable. An improvement on this method involves “propagating constraints” by sending valid values [10].

This protocol is applicable to self-interested agents since they each contribute decisions when the neediest. This protocol can be adapted for planning agents. The variables are shared activity parameters. The DCSP constraints are *equals* relations among agents sharing the activities. An agent’s protocol must keep track of a priority it assigns the shared activity, the priorities that the other sharing agents assign to the activity, and separate priorities for the agents themselves (for tie-breaking). These priorities can be parameters of the shared activity. A no-good message is a set of parameter constraints. Below are the protocol methods for updating permission constraints and rank and generating no-goods.

Asynchronous Weak Commitment modifyPermissions method

- if have highest priority
 - give self master permissions
- else
 - restrict self to slave permissions

Asynchronous Weak Commitment modifyConstraints method

- if cannot resolve local conflicts and conflicts with constraints of higher ranking agents

- set rank parameter of self to highest rank of sharing agents plus one
- generate no-good as a conjunction of other conflicting shared parameter values

The asynchronous weak commitment algorithm for DCSPs is shown to be sound and complete—the agents are guaranteed to converge to valid assignments of values to variables if they exist. In SHAC there is no guarantee of completeness (convergence). This is because SHAC does not restrict the planners to be complete. Since continual planning requires reactivity to state changes and failures, completeness is difficult to ensure in real-time. Future work is needed to determine how AWC (and other protocols) can be combined with complete planners to ensure convergence.

4 Protocol Evaluations

We evaluate four protocols (chaos, master/slave, round robin, and AWC) in an abstract domain where spacecraft propose joint measurements requiring specific capabilities. Each of the protocols are varied according to their use of constraints and the agents sharing the activities. The motivation is to see how well different argumentation strategies for cooperative and self-interested agents converge on consistent solutions according to time and communication overhead.

Protocols are varied to generate constraints or not. The protocols are further varied to update constraints every 1, 2, 5, or 10 SHAC cycles. In addition, they are alternatively updated only when the current plan is in conflict with constraints. The set of sharing agents alternatively includes all agents or just the originating agent and the agent providing the requested capability. We will refer to the former as a *broadcast* protocol since shared activity updates are broadcast to all others.

A joint measurement activity in our experimental domain requires some number of each type of capability and refines into concurrent activities, each consuming an agent's capability. The agents' capabilities each can only be used once at a time. A variation of the domain also restricts the agents to only be able to provide one capability at a time. The problem is to assign agents' capabilities to requested joint measurements while meeting these constraints. Solvable problems are randomly generated with 3 to 9 agents, 1 to 7 capability types, and 1 to 9 joint measurements, each requiring 1 to 4 of each capability type. This can result in a maximum of 252 shared activities.

SHAC interfaces with the ASPEN planning system [1]. ASPEN is a heuristic iterative repair planner that repeatedly chooses a flaw (or optimization criterion), chooses a repair method (such as add, move, or delete), chooses an activity, and applies the repair method to the activity to resolve the flaw (or improve utility). The choices are governed by built-in or user-defined heuristics. Because this local search approach keeps no backtracking states, the no-goods employed by AWC are not critical to solving problems. We implement AWC without no-goods, but our future work will evaluate strategies for managing them.

ASPEN's heuristics randomly choose agents to fulfill capabilities according to the agent's local restrictions and also in accordance with any SHAC constraints collected from other agents. If there are no valid agent assignments, a new agent will be chosen randomly with 10% probability. In another domain variation, an agent can reason about the local constraints on capabilities of other agents based on the activities visible to it.

Figure 3 shows the number of problems each solved within the time limit indicated on the x-axis for each protocol. For all protocol and domain variations, AWC outperforms the others with respect to CPU and actual clock time in converging on valid solutions. Chaos performs worse because it often returns inconsistent solutions due to the lack of causal consistency in message passing. Master/slave fails to perform as well because of its inability to explore the full search space. Figure 4 shows that

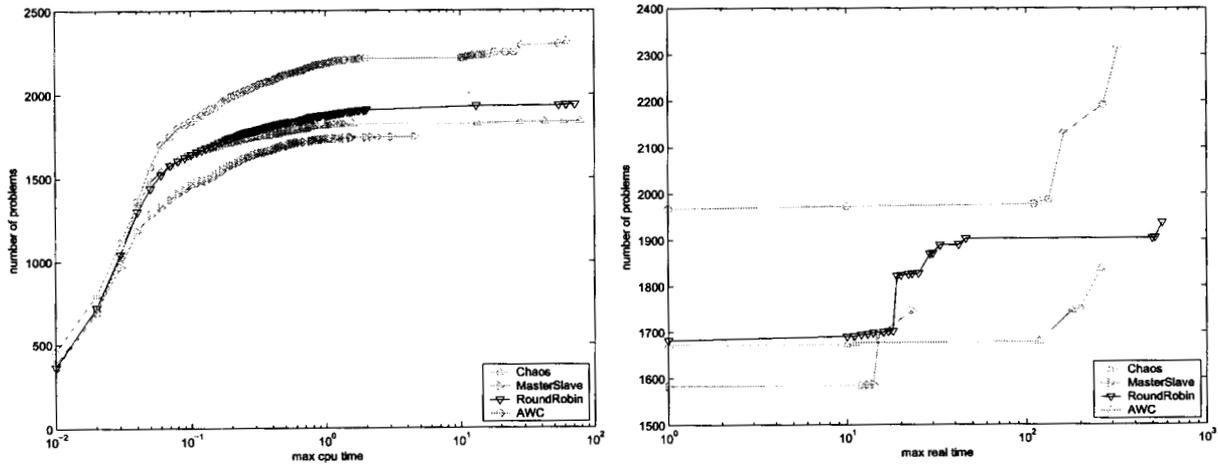


Figure 3: The overall number of problems solved for each protocol vs. time

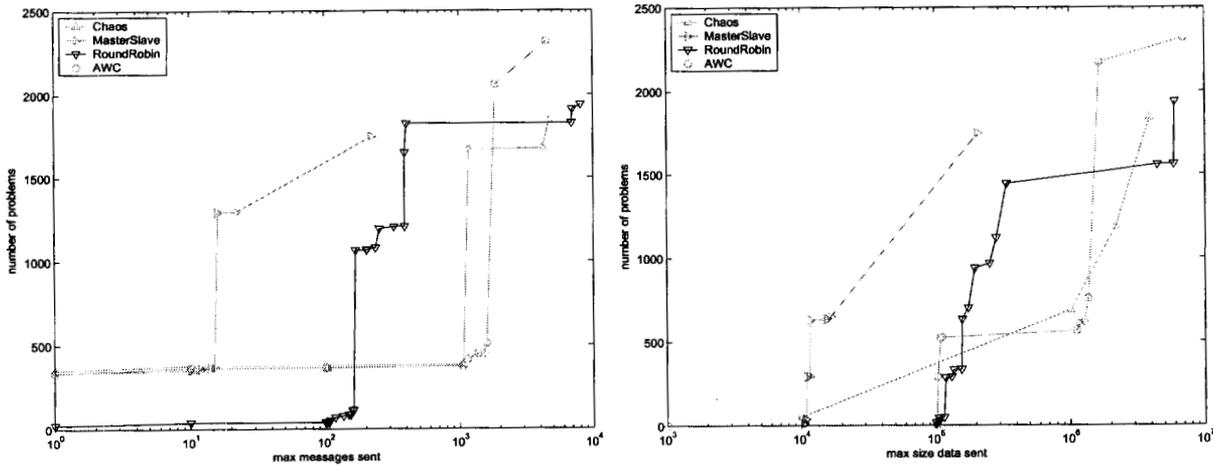


Figure 4: Problems solved for each protocol vs. communication overhead

master/slave has the lowest communication overhead. This is not surprising since only one agent can send updates for any shared activity. Round robin has significantly higher overhead for its 400 easiest problems but is almost an order-of-magnitude less costly than chaos and AWC for problems of medium difficulty. This abrupt swing is due to its large performance difference when broadcasting and not.

In Figure 5 we compare the protocols when all agents share the activities and when only those directly involved share. Here, we see that round robin outperforms all others when not broadcasting although similar to AWC. It is unclear why round robin performs so poorly when broadcasting. We speculate that the constant flood of incoming messages interferes with conflict resolution. AWC is also degraded by broadcasting, but chaos improves greatly because the inconsistencies are stamped out by the increased number of updates caused by broadcasting.

The incorporation of constraints has mixed results for the protocols as shown in Figure 6. The protocols compared here each use the sharing strategy that works best for it. Round robin and chaos perform better without constraints. This could again be caused by additional communication overwhelming conflict resolution. For chaos, the propagation of constraints actually hurts its ability to return consis-

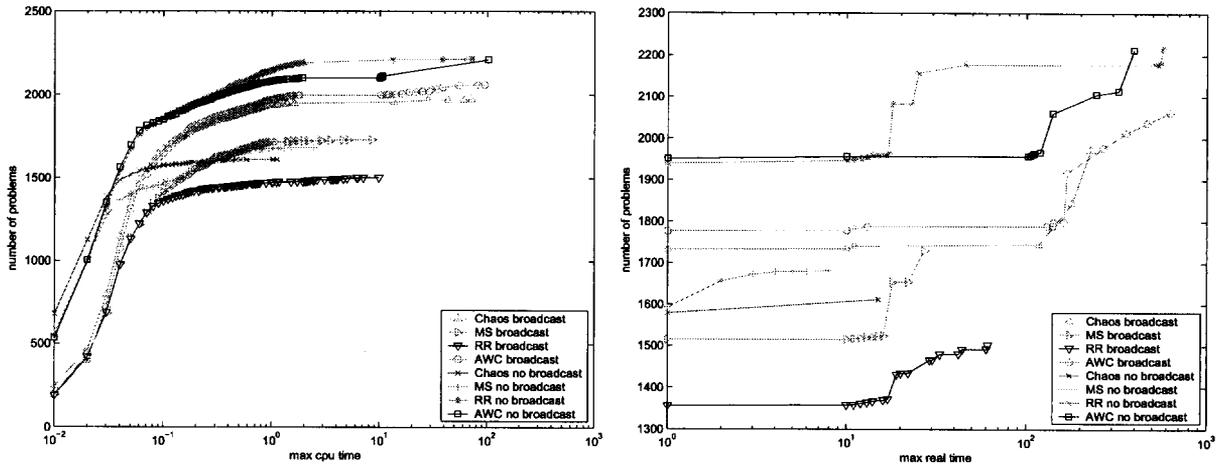


Figure 5: Problems solved for broadcast and limiting sharing protocol versions

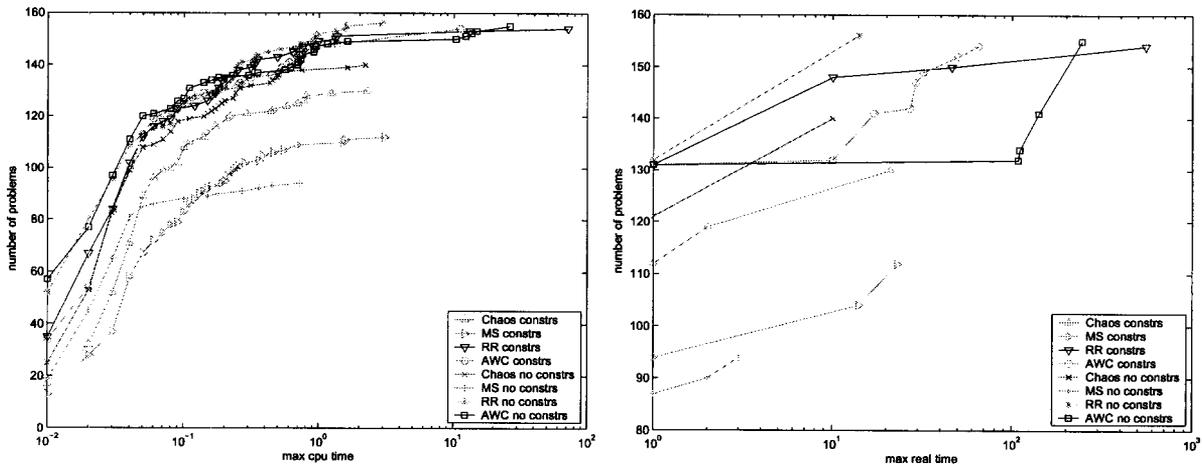


Figure 6: Problems solved with and without constraint propagation

tent solutions. Master/slave benefits from constraints because they allow the master to see the local constraints of slaves and cooperatively explore the global search space. AWC performs better on easier problems without constraints likely for the same reasons as round robin. However, for harder problems, constraints help AWC converge to solutions more quickly. This is consistent with DCSP results where propagating constraints improves performance because the agents are able to choose values that are more flexible with others. Thus, constraints enable more cooperative behavior, benefiting master/slave and AWC.

5 Discussion and Related Work

Conflicts among a group of agents can be avoided by reducing or eliminating interactions by localizing plan effects to particular agents [12], and by merging the individual plans of agents by introducing synchronization actions [8]. In fact, planning and merging can be interleaved [7]. Earlier work studied

interleaved planning and merging and decomposition in a distributed version of the NOAH planner [4] that focused on distributed problem solving. More recent research builds on these techniques by formalizing and reasoning about the plans of multiple agents at multiple levels of abstraction to localize interactions and prune unfruitful spaces during the search for coordinated global plans [3].

DSIPE [6] employs a centralized plan merging strategy for distributed planners for collaborative problem solving using human decision support. Like our approach, local and global views of planning problem help the planners coordinate the elaboration and repair of their plans. DSIPE provides insight into human involvement in the planning process as well as automatic information filtering for isolating necessary information to share. While our approach relies on the domain modeler to specify up front what information will be shared, SHAC supports a fully decentralized framework and focuses on interleaved coordination and execution.

In many ways this work is following the Generalized Partial Global Planning approach to using a mix of coordination protocols tailored for the domain [5]. SHAC offers an alternative framework for separating implementation of these mechanisms from the planning algorithms employed by specific agents. Unlike GPGP, SHAC provides a modular framework for combining lower-level mechanisms to create higher-level roles and protocols. Our future work will build on GPGP's evaluations of mechanism variations to better understand how agents should coordinate for domains varying in agent interaction, communication constraints, and computation limitations.

Finally, TEAMCORE provides a robust framework for developing and executing team plans [15, 14]. This work also offers a decision-theoretic approach to reducing communication within a collaborative framework. Research is needed to investigate the integration of coordinated planning with robust coordinated execution.

An assumption commonly made in multiagent research is that agents will be able to communicate at all times reliably. However, this is rarely the case with spacecraft that can only intermittently communicate and have significant communication delays (e.g. with Earth). Guaranteeing consensus on beliefs and intentions is impossible without certain communication guarantees [13]. Understanding the communication properties that make consensus possible and the overhead for establishing consensus is critical for multiagent research.

6 Conclusion

We introduced a distributed planning framework built on ideas of argumentation that is capable of continually coordinating planning agents. We described its capabilities and gave examples of argumentation mechanisms (protocols) built on these capabilities. We showed how variations of argumentation perform on joint measurement problems for interacting spacecraft. The round robin and AWC protocols generally avoid problems of consistency and limited search space encountered by simpler approaches. The round robin exhibited the best balance of time and communication costs when the number of sharing agents was minimized but performed similar to AWC for the hardest problems. For problems it can solve, the master/slave protocol uses the least communication overhead. The default unstructured SHAC protocol (chaos) shows promise if given causally consistent communication guarantees. Our future work is aimed at evaluating the benefits of other non-argumentation-based protocols with these for different classes of multiagent domains and examining real-time performance during execution.

References

- [1] S. Chien, G. Rabideu, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. Automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps*, 2000.
- [2] B. Clement and A. Barrett. Continual coordination of shared activities. In *Proc. AAMAS*, 2003.
- [3] B. Clement and E. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*, pages 213–227, 2000.
- [4] D. Corkill. Hierarchical planning in a distributed environment. In *Proc. IJCAI*, pages 168–175, 1979.
- [5] K. Decker. *Environment centered analysis and design of coordination mechanisms*. PhD thesis, University of Massachusetts, 1995.
- [6] M. desJardins and M. Wolverton. Coordinating a distributed planning system. *AI Magazine*, 20(4):45–53, 1999.
- [7] E. Ephrati and J. Rosenschein. Divide and conquer in multi-agent planning. In *Proc. AAAI*, pages 375–380, July 1994.
- [8] M. P. Georgeff. Communication and interaction in multiagent planning. In *Proc. AAAI*, pages 125–129, 1983.
- [9] B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–358, 1996.
- [10] H. Jung, M. Tambe, and S. Kulkarni. Argumentation as distributed constraint satisfaction: Applications and results. In *Proc. Intl. Conf. Autonomous Agents*, 2001.
- [11] S. Kraus, K. Sycara, and A. Evanchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–70, 1998.
- [12] A. Lansky. Localized search for controlling automated reasoning. In *Proc. DARPA Workshop on Innov. Approaches to Planning, Scheduling and Control*, pages 115–125, November 1990.
- [13] S. Mullender. *Distributed Systems*. Addison-Wesley New York, 1995.
- [14] D. Pynadath, M. Tambe, N. Cauvat, and L. Cavedon. Toward team-oriented programming. In *Proc. ATAL*, 1999.
- [15] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
- [16] M. Tambe and H. Jung. The benefits of arguing in a team. *AI Magazine*, 20(4), 1999.
- [17] M. Yokoo and K. Hirayama. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. on KDE*, 10(5):673–685, 1998.