Cooperative Negotiation for Soft Real-Time Distributed Resource Allocation *

Roger Mailler
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
mailler@cs.umass.edu

Victor Lesser
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
lesser@cs.umass.edu

Bryan Horling
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
bhorling@cs.umass.edu

ABSTRACT

In this paper we present a cooperative negotiation protocol that solves a distributed resource allocation problem while conforming to soft real-time constraints in a dynamic environment. Two central principles are used in this protocol that allow it to operate in constantly changing conditions. First, we frame the allocation problem as an optimization problem, similar to a Partial Constraint Satisfaction Problem (PCSP), and use relaxation techniques to derive conflict (constraint violation) free solutions. Second, by using overlapping mediated negotiations to conduct the search, we are able to prune large parts of the search space by using a form of arc-consistency. This allows the protocol to both quickly identify situations where the problem is over-constrained and to identify the appropriate fix to the over-constrained problem. From the global perspective, the protocol has a hill climbing behavior and because it was designed to work in dynamic environments, is an approximate one. We describe the domain which inspired the creation of this protocol, as well as discuss experimental results.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

*The effort represented in this paper has been sponsored by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-2-0525 and the National Science Foundation under grant number IIS-9812755. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), Air Force Research Laboratory, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

General Terms

Algorithms, Design, Experimentation

Keywords

Resource Allocation, Cooperative Negotiation, Distributed Problem Solving

1. INTRODUCTION

Resource allocation is a classic problem that has been studied for years by multi-agent systems researchers [1, 10]. The reason for this is that resource allocation shares a number of characteristics that are common to a wide range of multi-agent domains. For example, resource allocation requires search and is often too complex and time consuming to perform in a centralized manner when the environmental characteristics are both distributed and dynamic. In fact, in environments where search is being conducted and the costs associated with continuously centralizing alot of information are impractical, distributed techniques become imperative.

Negotiation, a form of distributed search [9], has been viewed as a viable technique for handling complex searches of this type that include multi-linked interacting subproblems [1]. Researchers in this domain have focused primarily on resource allocation scenarios that are formulated as distributed constraint satisfaction problems [8, 12]. In this work, we extend this classic formulation in two ways. First, we introduce soft real-time constraints on the protocol's behavior. These require the negotiation to adapt to the remaining available time, which is estimated dynamically as a result of emerging environmental conditions. Second, we reformulate the resource allocation task as an optimization problem, and like the Distributed Partial Constraint Satisfaction Problem (PCSP) [2, 3, 4], use constraint relaxation techniques to find a conflict-free solution while maximizing the social utility of the agents.

In this paper, we present a distributed, mediation-based negotiation protocol that takes advantage of the cooperative nature of the agents in the environment to maximize social utility. By mediation based, we are referring to the ability of each of the agents to act in a mediator capacity when resource conflicts are recognized. As a mediator, an agent gains a localized, partial view of the global allocation problem and makes suggestions to the allocations for each of the agents involved in the mediation. This allows the mediator to identify over-constrained subproblems and make suggestions to eliminate such conditions. In addition, the mediator can perform a localized arc-consistency check, which potentially allows large parts of the search space to be eliminated. Together with the fact that regions of mediation overlap, the agents rapidly converge on solutions that are in most cases good enough and fast enough. Overall, the protocol has many characteristics in common with distributed breakout [16], particularly its distributed hill-climbing nature and the ability to exploit parallelism by having multiple negotiations occur simultaneously.

In the remaining sections of this paper, we introduce the distributed monitoring and tracking application which motivated the development of our protocol. Next, we describe the Scalable Protocol for Anytime Multi-level negotiation (SPAM), a distributed, mediation-based negotiation protocol that was developed and has been tested on actual sensor hardware. In section 4, we will introduce Farm, a distributed simulation environment used to test SPAM, and in section 5, we present and discuss the results of testing SPAM within that simulator. The last section of the paper will present conclusions and future directions for this work.

2. DOMAIN

The resource allocation problem that motivated this work requires an efficient allocation of distributed sensing resources to the task of tracking targets in an environment. In this problem, multiple sensor platforms are distributed with varying orientations in a real time environment [6]. Each platform has three distinct radar-based sensors, each with a 120 degree viewable arc, which are capable of taking amplitude (measuring distance from the platform) and/or frequency (measuring the relative velocity of the target) measurements. In order to track a target, and therefore obtain utility, at least three of the sensor platforms must take coordinated measurements of the target, which are then fused to triangulate the target's position. Increasing the number, frequency and/or relative synchronization of the measurements yields better overall quality in estimating the target's location and provides a higher quality solution. The sensor platforms are restricted to only taking measurements from one sensor head at a time with each measurement taking about 500 milliseconds. These key restrictions form the basis of the resource allocation problem.

Each of the sensor platforms is controlled by a single agent which may take on multiple organizational roles, in addition to managing its local sensor resources. Each of the agents in the system maintains a high degree of local autonomy, being able to make trade-off decisions about competing tasks using the Soft Real Time Architecture [11].

One notable role that an agent may take on is that of track manager. As a track manager, the agent becomes responsible for determining which sensor platforms and which sensor heads are needed both now and in the future for tracking a single target. Track managers also act to fuse the measurements taken from the individual sensor platforms into a single location. Because of this, track managers are the focal point of any negotiation activities that take place as part of resolving resource contention.

Dynamics are introduced into the problem as a result of target movement. During the course of a run, targets will continuously enter and leave the viewable area of different sensors, which will then require track managers to continuously evaluate and revise their resource requirements. This, in turn, changes the underlying structure of the actual allocation problem. In addition, these dynamics drive the need for real-time negotiation, because a particular problem structure only holds for a limited amount of time.

Resource contention is introduced when more than one target enters the viewable range of the same sensor platform. Because of the time it takes to perform a measurement, combined with the fact that each sensor can take only one measurement at a time, track managers must come to an agreement over how to share sensor resources, without causing any targets to be lost. This local agreement can have profound global implications. For example, what if as part of its local agreement, a track manager relinquishes control of a sensor platform and takes another instead? This may introduce contention with another track manager already using that sensor, who may then have to request alternate sensor resources to make up for the new deficiency.

2.1 The Resource Allocation Problem

Generally speaking, we say that a resource allocation problem is the problem of assigning a (usually limited) number of resources to a set of tasks. Each of the tasks may have different resource requirements, and may have the potential for varying utility depending on which resources they use. The goal is to maximize the global utility of the assignment, choosing the right options for the tasks and assigning the correct resources to them. More formally, a resource allocation problem is comprised of:

- a set of tasks, $T = \{t_1, \cdots, t_n\}$
- a set of resources $R = \{r_{1,1}, \cdots, r_{j,k}\}$ where j is the number of resources and k is the planning horizon for the resource.
- a set of utility functions each associated with one of the tasks $U = \{U_1, \cdots, U_n | U_i : 2^R \mapsto \Re\}$

The goal of the problem is to come up with an allocation $A = \{a_1, \cdots, a_n | a_i \in 2^R\}$ such that $\sum_{i=1}^{n} U_i(a_i)$ is maximized and $\bigcap_{i=1}^{n} a_i = \emptyset$. The notation 2^R is used to indicate the power-set of the resources. Because the resource requirements may change over time, or a particular pattern of resource usage may be needed to obtain utility for a task, resources are broken down on both the resource and time dimensions, hence the need for a planning horizon. Increasing the number of resources or the planning horizon can have a significant effect on the overall complexity of the allocation problem, which is known to be NP-complete.

It should be noted that the utility of assigning a set resources to a task is strictly dependent on the utility function defined for the task. What also should be clear is that due to the sharing of resources, increasing the utility of a particular task may not increase the global utility. We make no assumptions in this paper about task independence.

The distributed version of the resource allocation problem, which is the focus of this paper, has each task assigned to a single agent. However, in general an agent may take on more than one task.

2.2 Tracking as Resource Allocation

Modeling the target tracking domain as a resource allocation problem is fairly straightforward. Each of the targets in the environment can be considered a task, which is assigned to a track manager. The sensors are the resources and the

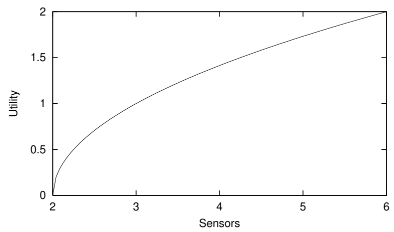

Figure 1: *Utility of taking a single, coordinated measurement from a set of sensors.*

job of the track managers is to obtain enough sensing time from the correct sensors to track their targets.

At any given time, each of the targets is within the viewable range of some subset of the sensors. That means that as the targets move from the viewable range of some sensors to others, the utility function associated with each of the tasks change. In addition, tracking involves coordinating measurements from three or more sensors which are then fused together to form an estimated position of the target. Increasing the number of sensors improves the quality of the estimate by the function given in figure 1. Increasing the measurements taken in a given period of time yields a linear increase in the overall quality of the track.

Because targets are often in the viewable range of a sensor for an extended period of time, planning within our system is periodic. This simply means that the sensors continuously repeat their assigned schedules until a change is made. We often refer to the planning horizon as a period and an individual element with the schedule as a slot.

If we say that M_s^i is the set of good sensors measurements (can see the target) leading to the positional estimate in a single slot s for a task i, then the utility function for that task during a specific period is:

$$U_i(a_i) = \sum_{s=1}^{k} Util(M_s^i)$$

The special nature of the utility functions in the tracking domain actually allow us to consider a much smaller subset of the possible allocations for a given task. In fact, track managers within our system use a simplified set of objective levels defined by their utility functions to assign resources to their targets. Each objective level is expressed as a cross product $D_m \times D_s$ denoting the number for resources from their acceptable set, desired for a number of slots in planning horizon. For example, a track manager may wish to have three sensors for two slots, which is denoted 3×2. Although the number of slots in a period is variable, for this domain, we typically set it to match the number of sensor heads on each platform, which is three.

Note that if a target is ignored (i.e. not being triangulated at all during a full period), we penalize ourselves by subtracting one from the social utility.

3. PROTOCOL

The Scalable Protocol for Anytime Multi-level (SPAM) negotiation is built around the principle of good enough, fast enough. As such, the protocol is actually divided into two major stages. As the protocol transitions from the first stage to the second, the agent acting as the track manager gains more context information and is, therefore, able to improve the quality of its overall decision. In addition, to allow stage 2 time to complete, without loosing all quality in the mean time, stage 1 of the protocol always ensures that at least some solution has been obtained. So, at any time after the completion of stage 1, the track manager can choose to stop the protocol and is ensured to have a solution, albeit not necessarily a good one (not optimal and not necessarily conflict free).

In the current implementation, the SPAM protocol is activated whenever a local change in the resources is needed (a new target, adding/removing resources, etc) or if the manager detects a conflict within one of the resources it's using. Resource needs in this domain are monitored and updated as the target moves through the environment. Conflicts are detected by the track managers through notification from the sensors. It is certainly conceivable and, in fact likely, for two conflicting managers to detect a conflict at the same time. We prevent two neighboring managers from mediating a negotiation by using a distributed locking technique.

3.1 Stage 1

Stage 1 of SPAM serves two primary functions. The first attempts to find a solution within the context of the information that the protocol has when it starts up. Like the Asynchronous Weak Commitment (AWC) protocol [15], each of the agents tries to find an assignment that is consistent with their potentially incomplete or inconsistent *agent_view*. However, because the protocol attempts to maximize the social utility, each of the agents tries to maximize their local utility without causing new constraint violations. If this can be done, then no further negotiation is necessary, and the protocol terminates at the end of stage 1.

We should mention that a trade-off that exists between communication overhead and utility, due to the initial selections of the objective level in stage 1. If each of the managers chooses to use every available resource (sensors able to see their target), the possibility for contention over resources greatly increases in the environment, thereby causing the execution of stage 2 to occur more frequently. However, if the agents decide to start with at a lower objective level (and correspondingly less utility), the social utility may suffer unnecessarily.

Currently, stage 1 has what we refer to as a concession rate. This rate defines how much of the local solution quality a track manager is willing to concede to find a violation-free solution, in an attempt to avoid the potentially expensive stage 2 negotiations. The rate is defined as a percentage of the manager's current utility, so as the manager's utility drops, the amount they are willing to concede drops as well. That means that in critically constrained tracking environments, the managers attempt to negotiate more frequently.

The second function of stage 1 is to ensure some utility is obtained while waiting for stage 2 to complete. If a violation-free solution cannot be obtained during stage 1 of the protocol, one of two things will happen. If the reason the protocol was started was a resource requirement change, a temporary solution is applied to the problem. This solution, although not conflict free, has the ability to obtain at least some utility while the manager tries to get a better solution. Conflicts that are unresolved are actually left to the individual sensor agents to handle. Sensor agents can use one of a number of techniques, including slot boundary shifting, less expensive measurement types, or task rotation, in or-

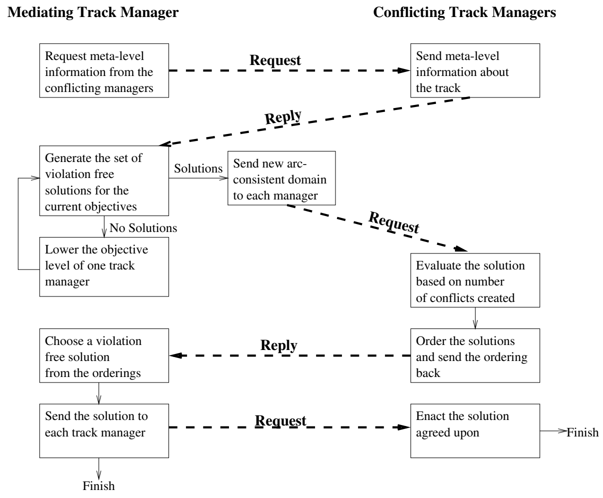

Figure 2: *Stage 2 of the SPAM negotiation protocol.*

der to solve such conflicts. To the track manager, whether or not they get a measurement from a conflicted sensors is probabilistically random.

Temporarily applied solutions do not use the concession rate. In fact, because of environmental changes and the probabilistic nature of getting measurements from conflicted sensors, managers always use their maximum possible objective level (within the bounds of the number of sensors that can see the target). The first reason is rather subtle, but important. Let's say that a new resource were added to the possible resources that could be used by manager T1. Let's also say that another manager, T2, who has more than enough available resources to itself, were using that entire resource. If T1 starts a negotiation at that lower level, it can never obtain its highest level through the negotiation, even though a solution exists where T2 just gives up the entire conflicted resource. From a PCSP perspective this just means that if the structure of the CSP changes, the PCSP attempts to satisfy all of the constraints before relaxing any of them.

If stage 1 was activated because of a newly discovered conflict, and a conflict-free solution cannot be found, then the manager just enters stage 2. It does not concede, does not bind a temporary solution, and it does not reset its objective level. This case most often occurs when there is a multi-linked problem within the environment. Let's say you have three managers, T1, T2, and T3. T1 has a conflict with T2, and T2 is sharing resources with T3, but is not in conflict. As a result of a negotiation between T1 and T2 their conflict is solved, but it creates a conflict between T2 and T3. When T2 recognizes this problem, if it reset its objective level, the problem becomes harder to solve because it may reintroduce conflict with T1 as well as increasing the conflict with T3.

3.2 Stage 2

Stage 2 of SPAM is the heart of the negotiation protocol (See figure 2). Stage 2 attempts to resolve all local conflicts that a track manager has by elevating the negotiation to the track managers that are in direct conflict over the desired resources. To do this, the originating track manager takes the role of the negotiation mediator for the local conflict (note that multiple negotiations can occur in parallel in the environment). As the mediator, it becomes responsible for gathering all of the information needed to generate alternative solutions, generating these solutions which may involve changes to the objective levels of the managers involved, and finally choosing a solution to apply to the problem. Because the solutions are generated without full global information, however, the final solution may lead to newly introduced non-local conflict. If this occurs, each of the track managers can choose to propagate the negotiation with itself as the mediator in order to resolve this conflict if they have the time. So, what started out as a new target or resource requirement change, may lead to the negotiation propagating across the problem landscape.

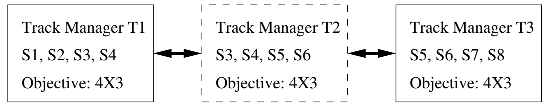

Figure 3: *Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.*

Looking at this from a more formal perspective. If the set of resources that are usable for a single task t_i is defined as

$$R(t_i) = \{r_{u,v} | r_{u,v} \in R \wedge \exists a (U_i(a \cup r_{u,v}) > U_i(a))\}$$

then the set of acceptable resource assignments for a single task t_i is

$$D(t_i) = \{a | a \in 2^{R(t_i)} \wedge U_i(a) > 0\}$$

and the neighbor tasks to a mediator m are

$$N_m = \{t_i | t_i \in T \wedge R(t_m) \cap R(t_i) \neq \emptyset\}$$

then the problem that a mediating manager m is working on is

- a set of tasks, $T_m = \{t_m \cup N_m\}$
- a set of resources $R_m = \{r_{u,v} | r_{u,v} \in (\bigcup_{\forall t_i \in N_m} R(t_i)) \cap R(t_m)\}$
- a set of utility functions $\hat{U} = \{\hat{U}_i | t_i \in T_m\}$

The goal of this subproblem is the same as the goal of the global problem. The notation $\hat{U}_i$ is used to indicate an approximation function to the actual U_i for each of the managers. Also note that $R_m \subseteq \bigcup_{\forall t_i \in N_m} R(t_i)$. What this means is that the view of the mediating manager is limited to only the constraints that arise from the sharing of a resource with the mediator. These conditions, when combined together, indicate that the estimated utility of a solution to the subproblem is always either equal to or an over-approximation to the actual utility obtained socially. This is simply a by-product of performing a localized search. The mediator never knows if the assignments it proposes at a given utility value will cause conflict outside of its view, which is why we allow the managers to propagate. The best way to explain how stage 2 operates is through an example. Consider figure 3. This figure depicts a commonly encountered form of contention. Here, track manager T2 has just been assigned a target. The target is located between two existing targets that are being tracked by track managers T1 and T3. This creates contention for sensors S3, S4, S5, and S6.

Following the protocol for the example in figure 3, track manager T2, as the originator of the conflict, takes on the role of negotiation mediator. It begins the solution generation phase by requesting meta-level information from all of the track managers that are involved in the resource conflict. The information that is returned includes the current objective level that the track manager is using, the number of sensors which could possibly track the target, the names of the sensors that are in direct conflict with the mediator, and any additional conflicts that the manager has. To continue our example, T2 sends a request for information to T1 and T3. T1 and T3 both return that they have 4 sensors that can track their targets, the list of sensors that are in direct conflict (i.e $T1(S_3, S_4)$, $T3(S_5, S_6)$) their objective level (4×3 for both of them) and that they have no additional conflicts outside of the immediate one being considered.

Using this information, T2 is able to generate $D(t_i)$ for each of the tasks in the set T_m for the objective levels that are passed in as part of the meta-level information(see section 3.3). With the full set of $D(t_i)$'s, it's fairly easy to generate all possible satisfying assignments $\mathbf{A}$ with each element being a particular $A_m = \{a_i | t_i \in T_m \wedge a_i \in D(t_i)\}$ s.t. the condition $\bigcap_{\forall a_i \in A_m} a_i = \emptyset$ is met.

As you can see in figure 2, T2 enters a loop that involves attempting to generate these sets followed by lowering one of the track manager's objective level if $\mathbf{A} = \emptyset$ given the current objective levels of each of the track managers. One of the principle questions that we are currently investigating is how to choose the track manager that gets its objective level lowered when $\mathbf{A}$ is empty. Right now, this is done by choosing the track manager with the highest current objective level, which cannot support its demands with resources outside of the set R_m and lowering them. This has the overall effect of balancing the objective levels of the track managers involved in the negotiation. Whenever two or more managers have the same highest objective level, we choose to lower the objective level of the manager with the least amount of external conflict. By doing this, it is our belief, that track managers with more external conflict will maintain higher objective levels, which leaves them more leverage to use in subsequent negotiations as a result of propagation.

You should note that although this has similarities to the techniques used in PCSPs, this differs in that the actual CSP problem changes as the objective levels are changed. PCSP techniques, such as [2, 3, 4] choose a subset of the constraints which are satisfiable, we change the structure of the constraints until it is satisfiable. We also differ from the Distributed Constraint Optimization (DCOP) [13] work in that although they have a utility function over the possible assignments to a problem, they do not change the underlying CSP to ensure it is satisfiable.

The solution generation loop is terminated under one of two conditions. First, if given the current objective levels for each of the track mangers, the set $\mathbf{A} \neq \emptyset$, the negotiation enters the solution evaluation phase. Second, we cannot find a track manager to lower without $D(t_i) = \emptyset$ and $\mathbf{A} = \emptyset$. Under this condition, the negotiation session is terminated and the mediator takes a partial solution at the lowest objective level that minimizes the resulting conflict, conceding that it cannot find a full solution.

Continuing our example, T2 first lowers the objective level of T1 (choosing T1 at random because they all have equal external conflict). No full solutions are possible under the new of set objective levels, so the loop continues. It continues, in fact, until each of the track managers has an objective level of 3×2 at which time T2 is able generate a set of 216 (the number of elements in $\mathbf{A}$) solutions to the problem.

During the solution evaluation phase, the mediator sends each of the track managers a set:

$$d_i = \{a | a \in D(t_i) \wedge \exists A_m \in \mathbf{A}(a \in A_m)\}$$

What should be clear is that each of the set d_i is arc-consistent for every constraint between elements in the set R_m. What that means is that for the mediator's resources, all constraints are satisfied.

Each track manager, using its set d_i and a revised objective level, can then determine which, if any, of the solutions are satisfiable given the local *agent_view* and which is best given the actual U_i. In our example, T2 sends 24 alternatives to T1, 24 alternatives to itself, and 24 alternatives to T3. In our current implementation, each of the track managers orders alternatives from best to worst based on the number of new conflicts that will be introduced and the desirability of the particular resources present in the alternative. This has a min-conflict heuristic [7] like flavor and is an integral part of the hill-climbing nature of the algorithm. Currently, we are looking at a number of alternative techniques for providing local preference information to the mediator including simply returning utility values for each solution and assigning solutions to a finite set of equivalence classes.

Once the mediator has the orderings from the track managers, it chooses a particular A_m to apply to the problem. This is done using a dynamic priority method based on the number of constraints each of the managers has external to the mediation, a form of meta-level information. The basic notion is similar to the priority order changes in AWC [15]; try to find the task which is most heavily constrained and elevate it in the orders. Our impression is that this helps stem the propagation of the negotiation because it leaves the most constrained tasks with the best choices. This allows those managers to maintain violation free solutions if they exist in the alternatives presented to them. Let's say that the priority ordering for the tasks is $(t_h, t_{h-1}, \cdots, t_0)$, etc. The mediator iteratively prunes the set $\mathbf{A}$ by creating a set $\mathbf{A}_{t_h} = \{A_m | A_m \in \mathbf{A} \wedge \forall A_i \in \mathbf{A}(priority_h(a_u \in A_m) \geq priority_h(a_v \in A_i))\}$. This newly created list is pruned in the same way for each of the managers until $|\mathbf{A}| = 1$.

In our example, T2 collects the ordering from T1, T2, and T3. T3 is given first choice. By its ordering it ranked alternative 0 the highest. This restricts the choice for T2 to alternatives 0, 1, 2, and 3. T2 ranked 0 highest from this set of alternatives, restricting T1's choice to its 0th, 1st, and 2nd alternatives. It turns out that T1 likes its 0th solution the best so the final solution is composed of T3's alternative 0, T2's alternative 0, and T1's alternative 0.

The last phase of the protocol is the solution implementation phase. Here, the mediator simply informs each of the track managers of its final choice. Each of the track managers then implements the final solution. At this point, each of the track managers is free to propagate and mediate a negotiation if it chooses to. Figure 4 shows the configuration of the sensors before and after T2 completes stage 2.

Because the SPAM protocol operates in a local manner, a condition known as oscillation can occur. Say that, from our previous example, track manager T1 originated a negotiation with track manager T2. In addition assume that T2 had previously resolved a conflict with manager T3, that ter-

	Slot 1	Slot 2	Slot 3
S1	T1	T1	T1
S2	T1	T1	T1
S3	T1/T2	T1/T2	T1/T2
S4	T1/T2	T1/T2	T1/T2
S5	T3/T2	T3/T2	T3/T2
S6	T3/T2	T3/T2	T3/T2
S7	T3	T3	T3
S8	T3	T3	T3

	Slot 1	Slot 2	Slot 3
S1		T1	T1
S2		T1	T1
S3	T2		T1
S4	T2	T1	T2
S5	T2		T2
S6	T3	T3	T2
S7	T3	T3	
S8	T3	T3	

Figure 4: *A solution derived by SPAM to the problem in figure 3. The table on the left is before track manager T2 negotiates with T1 and T3. The table on the left is the result of stage 2 negotiation.*

minated with neither T2 or T3 having unresolved conflict. Now when T1 negotiates with T2, T1 in the end gets a locally unconflicted solution, but in order for that to occur, T2 conflicts with T3. It is possible that when T2 propagates the negotiation, that the original conflict between T1 and T2 is reintroduced, leading to an oscillation.

There are actually a number of ways to prevent this from happening when the problem being worked on is static. For example, in both [14, 15], the authors use global prioritization, static in one, dynamic in the other, to prevent loops in the constraint network, and also maintain *nogood* lists to ensure a complete search.

We explored a method in which each track manager maintains a history of the sensor schedules that were being negotiated over whenever a negotiation terminated. By doing this, managers were able determine if they have previously been in a state which caused them to propagate a negotiation in the past. To stop the oscillation, the propagating manager lowered its objective level to force itself to explore different areas of the solution space. It should be noted that in certain cases oscillation was incorrectly detected using this technique, which resulted in having the track manager unnecessarily lower its objective level.

This technique is similar to that applied in [8], where a *nogood* is annotated with the state of the agent storing it. Unfortunately, none of these techniques work well when complex interrelationships exist and are dynamically changing. Because the problem changes continuously, previously explored parts of the search space need to be constantly revisited to ensure that an invalid solution has not recently become valid. Currently, we allow the agents to enter potential oscillation, maintaining no prior state other than objective levels, from negotiation to negotiation and rely on the environment to break oscillations. We consider the problem of oscillation prevention and detection in dynamic environments to be an open research question.

3.3 Generating Solutions

Generating the set **A** for the domain described earlier involves taking the information that was provided through communications with the conflicting track managers and assuming that the sensors that are in the set $\bigcup_{\forall t_i \in N_m} D(t_i) - R(t_m)$ are freely available. In addition, because the track manager that is generating full solutions only knows about the sensors which are in direct conflict, it only creates and poses solutions for those sensors. That means that $\forall a \in d_i (a \in R_m)$. The formula below illustrates the basic mechanism that task manager's use to generate task alternatives. Here, k is the number of slots that are available in the planning horizon, D_s is the number of slots that are desired based on the objective level for the track manager, $|R(t_i)|$ is the number of sensors available to track the target (those that can see it), D_m is the number of sensors desired in the objective function, and finally $C_i = |R(t_i) \cap R(t_m)|$ is the number of sensors under direct consideration because they are conflicting.

$$|D(t_i)| = \binom{k}{D_s} \left(\sum_{u=max(0, D_m - |R(t_i)| + C_i)}^{min(C_i, D_m)} \binom{C_i}{u} \right)^{D_s}$$

As can be seen by this formula, every combination of slots that meets the objective level is created, and for each of the slots, every combination of the conflicted sensors is generated such that the track manager has the capability of meeting its objective level using the sensors that are available. For instance, let's say that a track manager has four sensors S1, S2, S3, and S4 available to it. The track manager has a current objective level of 3×2 and sensors S2 and S3 are under conflict. The generation process would create the 3 combinations of slot possibilities and then for each possible slot, it would generate the combination of sensors such that three sensors could be obtained. The only possible sensor combinations in this scenario would be that the track manager gets either S2 or S3 (assuming that the manager will take the other two available sensors) or it gets S2 and S3 (assuming it only takes one of the other two). Therefore, a total of 27 possible solutions would be generated.

It is interesting to note that we use this same formula for alternative solutions in stage 1 of the protocol. This special case generation is actually done by simply setting $C_i = |R(t_i)|$. In this case, the formula above reduces to:

$$|D(t_i)| = \binom{k}{D_s} \binom{C_i}{D_m}^{D_s}$$

We can also generate partial solutions when there are a number of pre-existing constraints on the use of certain slot/sensor combinations. Simply by calculating the number of available sensors for each of the slots, and using this as a basis for determining which slots can still be used, we can reduce the number of possible solutions considerably.

Using the ability to impose constraints on the alternatives generated for a given track manager allows us to generate full solutions for the track managers in stage 2. By recursively going through the track managers using the results from earlier track managers as constraints for lower precedence ones, we can do a full search of the localized subproblem

You can view this as a tree-based search where the top level of the tree is the set of alternatives for one track manager. Each of the nodes at this level may or may not have a number of children which are the alternatives available to the second track manager and so on. Only branches of the tree that have a depth equal to one less than the number of track managers are added to the set **A**. If there are no branches that meet this criteria, then the problem is considered over constrained.

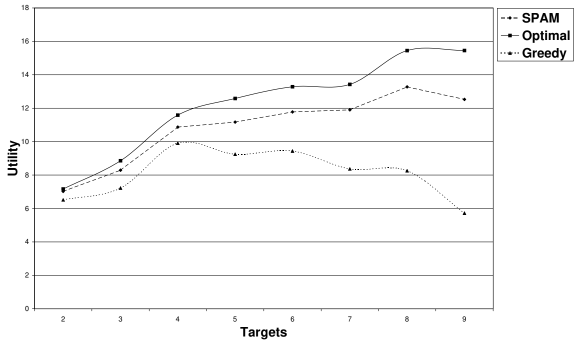

Figure 5: *Average utility of SPAM versus Greedy and Optimal allocation.*

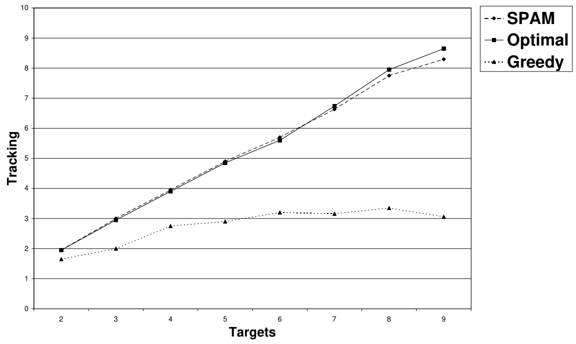

Figure 6: *Average number of targets tracked by SPAM versus Greedy and Optimal.*

4. SIMULATION

To test the SPAM protocol, we implemented a model of the domain described in section 2 in a simulation environment called Farm [5]. Farm is a component-based, distributed simulation environment written in Java where individual components have responsibility for particular encapsulated aspects of the simulation. For example, they may consist of agent clusters, visualization or analysis tools, environmental or scenario drivers, or provide some other utility or autonomous functionality. These components or agent clusters may be distributed across multiple servers to exploit parallelism, avoid memory bottlenecks, or utilize local resources.

The actual model used to test the SPAM protocol has both sensor and track manager agents. Each of the sensor agents represents a single sensor which was randomly placed in the world. These sensors agents are very simple, and only maintain a local schedule, which is not actually performed in any tangible way. A fixed number of targets is introduced into the world, and one track manager per target is created to manage the resources needed to track that target. The targets move through the environment with random trajectories that have a random, bounded speed. As the simulation progresses, the simulator continuously updates the position of the targets, and for each target calculates the set of sensors that are able to track it. The track managers can obtain their candidate sensor lists from the simulation environment and follow the SPAM protocol to allocate resources.

For comparison purposes, we have also implemented a Greedy and an Optimal track manager agent. Greedy agents work by requesting all of the available (can see their target) resources to track their target, potentially overriding each other in the sensors' schedules. The optimal agent computes the maximal set of objective levels that is satisfiable in the environment. This is done by having the optimal agent perform a complete search of the space of allowable objective levels, where each one is checked for satisfiability using a modified version of the complete search algorithm presented in section 3.3. To make this search go faster, we prevent the search from checking satisfiability on solutions that have utilities less than the best already obtained, and do a simple arc-consistency (using the pigeon hole principle) check to prune obviously over-constrained problems.

5. RESULTS

A total of 160 simulation runs were performed, using 20 sensors and 2 to 9 concurrent targets. The targets were placed in the environment and maintained a static location. We compared each of the approaches based on their achieved utility and the number of targets they tracked. The targets were not moved during the course of the run which allowed us to test the convergence rate of SPAM and give enough time for the optimal agent to compute its solution.

Figures 5, 6, and 7 summarize the results of the test. As you can see from the graph, SPAM does quite well when compared to both greedy and optimal. For the greedy method, the problem begins to become over-constrained at around 4 targets. SPAM provide reasonably good results (over 80% optimal for utility) for all of the configurations tested. Two things in particular are interesting about these results. First, for tracking targets, SPAM performs nearly 100% optimal. For the sake of these tests, a target is considered tracked if at least one triangulating measurement occurs with a given planning horizon. In some cases, SPAM actually tracked more targets than optimal. This is caused by the fact that SPAM is trying to optimize the balance of resources so that as many targets can be tracked as possible, optimal is just computing the best set of objective levels to maximize the social utility. It turns out that in many of the critically conflicted problems, optimal chose to ignore certain targets in order to track other better.

Figure 7 shows another interesting result. As the problem gets harder SPAM has a linear increase in the time it takes to converge. This is very promising, considering the allocation problem is known to be NP-complete. Unfortunately, we have not yet implemented other solutions, which could be used to compare this running time. We can, however, say that our practical experience using the protocol in a real-world sensor network [6] makes us believe that the techniques presented in this paper are applicable to real problems. It should also be noted that the optimal solution took between a few seconds (for two targets) to an entire day (for eight targets) to compute.

Something we were not able to show in the graphs is that there are cases when the greedy agent obtained higher utility than SPAM, but was ignoring a large number of the targets in order to achieve it. We think that this may be caused by not penalizing enough for ignoring targets. It is not clear what that penalty should be, and initially seems to be strongly domain dependent.

Lastly, there was at least one case were SPAM entered an

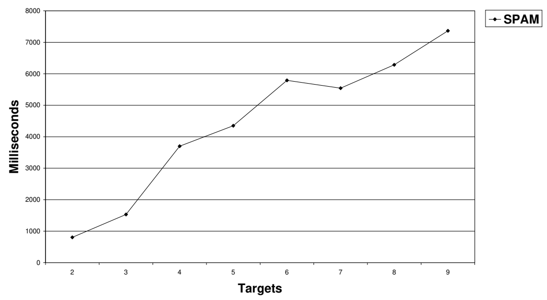

Figure 7: *Average time till SPAM reaches quiescence.*

oscillation. The utility obtained during the oscillation varied only slightly and the number of unresolved global conflicts fluctuated back and forth from 2 to 3. As mentioned earlier, this is a result of the localization of the search and in a dynamic environment may have been eliminated as the problem changed.

6. CONCLUSION

In this paper, we have described a distributed, mediation-based negotiation protocol which was built to solve resource allocation problems in a soft real-time environment. The protocol exploits the fact that agents within the environment are both cooperative and autonomous, and employs a number of techniques to operate in highly dynamic environments. Included in these techniques are mapping the resource allocation problem into an optimization problem and applying arc-consistency techniques to quickly prune the search space.

The results of this work are encouraging, and although we consider the problems associated with DCSPs and distributed resource allocation in dynamic environments to be an open research question, we feel that SPAM is a step in the right direction.

Future work on the protocol includes generalizing it to work in other environments. This includes adapting the search techniques to work when stricter, temporal constraints are placed on the utilization of resources and making the local search anytime. We are also looking at how to apply the techniques SPAM utilizes to work in static environments. Our hope is that mediation-based negotiation can be adapted to perform complete, distributed searches and that the convergence rate will remain quite fast compared to more conventional techniques.

7. ACKNOWLEDGMENTS

Thanks to Tim Middlekoop, Jiaying Shen, and Regis Vincent for helping during the initial protocol development.

8. REFERENCES

[1] S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), Nov. 1991.

[2] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3):21–70, 1992.

[3] K. Hirayama and M. Yokoo. Distributed partial constraint satisfaction problem. In G. Smolka, editor, *Principles and Practice of Constraint Programming (CP-97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 222–236. Springer-Verlag, 1997.

[4] K. Hirayama and M. Yokoo. An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *International Conference on Multi-Agent Systems (ICMAS)*, 2000.

[5] B. Horling, R. Mailler, and V. Lesser. Farm: A scalable environment for multi-agent development and evaluation. *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003)*, May 2003.

[6] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 417–424, 2001.

[7] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.

[8] P. J. Modi, H. Jung, M. Tambe, W.-M. Shen, and S. Kulkarni. Dynamic distributed resource allocation: A distributed constraint satisfaction approach. In J.-J. Meyer and M. Tambe, editors, *Pre-proceedings of the Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 181–193, 2001.

[9] T. Moehlman, V. Lesser, and B. Buteau. Decentralized negotiation: An approach to the distributed planning problem. *Group Decision and Negotiation*, 1(2):161–192, 1992.

[10] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.

[11] R. Vincent, B. Horling, V. Lesser, and T. Wagner. Implementing soft real-time agent control. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 355–362, 2001.

[12] M. Yokoo. *Distributed Constraint Satisfaction*. Springer Series on Agent Technology. Springer, 1998.

[13] M. Yokoo and E. H. Durfee. Distributed constraint optimization as a formal model of partially adversarial cooperation. Technical Report CSE-TR-101-91, University of Michigan, Ann Arbor, MI 48109, 1991.

[14] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.

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

[16] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *International Conference on Multi-Agent Systems (ICMAS)*, 1996.