A Cooperative Mediation-Based Protocol for Dynamic, Distributed Resource Allocation

Roger Mailler and Victor Lesser

***Abstract*— In this article, we present a cooperative mediation-based 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 mediation sessions 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 when the problem is over-constrained and to determine the appropriate repair. 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.**

I. INTRODUCTION

Resource allocation is a classic problem that has been studied for years by multi-agent systems researchers [1], [2]. 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 a lot of information are impractical, distributed techniques become imperative.

Cooperative, iterative search (negotiation), has been viewed as a viable technique for handling complex searches of this type that include multi-linked interacting subproblems [1]. Unfortunately, a common drawback to this technique is that it prevents the agents from making informed decisions about the effects of changing their local allocation without actually doing it. Because of the length of time required for this technique to converge on a solution, researchers have often abandoned the optimization portion of resource allocation, instead modeling them as distributed constraint satisfaction problems [3], [4], in order to provide reasonable solution speed.

In this work, we extend the traditional formulation of the resource allocation problem in two ways. First, we introduce soft real-time deadlines on the protocol's behavior. These deadlines require the protocol 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) [5]–[7], use constraint relaxation techniques to find a conflict-free solution while maximizing the social utility of the agents.

In this article, we present a distributed, mediation-based 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 session*. 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 without having to go through an number of trial and error steps. 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 [8], particularly its distributed hill-climbing nature and the ability to exploit parallelism by having multiple mediated sessions occur simultaneously.

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

II. 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 [9]. 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

Manuscript received January 30, 2004
R. Mailler and V. Lesser are with the University of Massachusetts

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 a coordinated measurement 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 one or more 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 SRTA agent architecture [10].

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 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 continuously enter and leave the viewable area of different sensors, which 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 problem solving, 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.

A. 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:

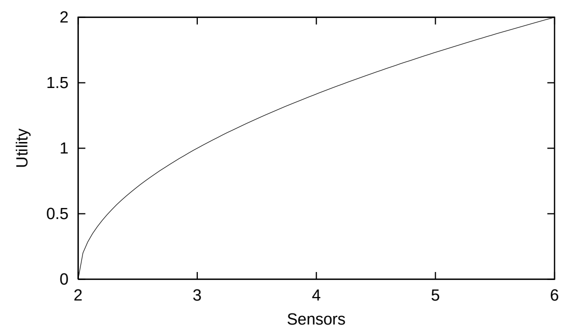


Fig. 1. Utility of taking a single, coordinated measurement from a set of sensors.

- 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 the following conditions are met:

1) $\sum_{i=1}^{n} U_i(a_i)$ is maximized.
2) $\bigcap_{i=1}^{n} a_i = \emptyset$.

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.

The first condition above basically makes this problem an optimization problem and can be viewed as a soft constraint on the solution. The second condition is a hard constraint since we know that a single resource cannot be applied to two tasks simultaneously. As we will discuss later, we may not always strictly adhere to the second condition using high level distributed search. In fact, we rely on the agents within the system to always ensure this condition is satisfied during times when the SPAM protocol has not.

According to this problem formulation, each task has its own utility function and the utility of assigning a set resources to a task is strictly dependent on that individual function. In fact, in dynamic domains, these function may change over time which alters the underlying relationships between tasks. What this also means 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.

B. 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 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 an estimate by the function given in figure 1 which is based on the RMS minimization method used to triangulate the targets. Increasing the frequency of the triangulation yields a linear increase in the overall quality of the track i.e. two measurements during a given period is twice as good as one.

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 (corresponds to k) as a period and an individual element within the period 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:

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

which basically says that the utility of a task for a specific period is the sum of the slot utilities for the slots within the period.

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 of 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.

There are essentially two benefits to using this abstract approach to resource scheduling. First, this representation vastly reduces the search space by discretizing time into slots and aligning the slots between the agents (the agents are time synchronized using Network Time Protocol (NTP)). It is easy for a manager to know that its measurements are coordinated if it has scheduled all of them during slot 1 and it knows that each of the sensors executes methods in slot 1 at about the same time. Second, by working abstractly, managers leave the details of the actual implementation of the period-based schedule to the agents themselves. This leaves the individual agents quite a bit of flexibility in how they internally manage competing tasks.

Note that if a target is ignored (i.e. not being triangulated at all during a full period), we penalize ourselves by subtracting two from the social utility. This penalty approximates the expected gain the agents would obtain by starving one of the track managers which makes the allocations a bit more “fair”.

III. Protocol

The Scalable Protocol for Anytime Mediation (SPAM) 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 interim, 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.

The SPAM protocol is activated whenever a local change in the resources is needed or if a manager detects a change in the level of contention within one of the resources it’s using. Detecting a change in the resource needs is done by monitoring the location of the target as it moves within the sensor field. Track managers constantly evaluate each of the sensors based first, on the ability of the sensor to see their target and second, on the expected value of the raw data returned by the sensor. This can be most easily understood as a change to the utility function used for the track. Whenever a target moves out of the view of one of the sensors currently being used to track it, into the view of a new sensor, or if a sensor currently not being utilized becomes more valuable than a sensor being used, the manager starts a SPAM session.

Whenever SPAM is activated for this reason, managers set their objective level to the highest possible value which ensures the hill-climbing nature of the algorithm. To understand this point, consider the following example:

Let’s say that a new resource were added to the possible resources that could be used by manager T1 to track its target. Let’s also say that another manager, T2, who has more than enough available resources to itself, was using that resource. If T1 starts SPAM at a low objective level, it may find a solution that is conflict-free, but will never realize that it could have gotten a better solution where T2 just gives up the entire conflicted resource. From a PCSP perspective this just means that whenever the structure of a CSP changes, the PCSP algorithm should reset its initial bound and attempt to satisfy all of the constraints before beginning relaxation.

The second reason for starting a SPAM session is when a manager recognizes a change in the level of contention within one of the sensors currently being used to track its target. This differs considerably from a resource change. Here the manager is recognizing that there is a change in the utility being obtained or could potentially be obtained for its target. The utility function itself has not been changed, only the interactions between it and some other function within another manager. Track managers detect these changes by monitoring the resource schedules of the sensors. To facilitate this process, sensors inform the managers that are utilizing them about the state of their resource schedule whenever a change occurs. It is certainly conceivable and, in fact likely, for two managers to detect changes in contention at the same time.

There are actually two separate cases here. When a manager recognizes a previously unknown conflict (i.e. a new restriction), it is most likely caused by another manager choosing an assignment that uses the resources because it either didn't know it was being used, or was forced to as a result of a mediation. In other words, this case most often occurs when there is a multi-linked problem within the environment. When managers recognize this case, they do not change their objective level before starting SPAM. The reason for this is easy to understand after considering an example. 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 mediation 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.

The other type of change occurs when there is a relaxation of contention on a resource. Again, managers recognize this type of change by monitoring the resource schedules of the sensors they are using. Whenever a manager realizes that it can improve its local utility (increase its objective level) without creating new conflicts, it sets its objective level to that new increased value, and starts SPAM. As you will see, when SPAM executes, it will make a simple local change to its assignment to take advantage of the additional resources because it can find a local solution that is conflict free.

A. Stage 1

Stage 1 of SPAM (see figure 2) serves three primary functions. The first is to find a suitable solution within the context of the information that the protocol has when it starts up. Like the Asynchronous Weak Commitment (AWC) protocol [11], each of the agents tries to find a resource assignment that is based solely on their incomplete or inconsistent view of the current resource schedules from the sensors. Resource assignments derived at this level must meet two criteria. First, they must find at the objective level that was set at the startup of SPAM. This ensures the continued hill-climbing nature of the search. Second, the resource assignment cannot create a conflict with another track manager. This criteria ensures that the overall effect of the assignment is not globally negative. If an assignment is found that adheres to these restrictions, then no further work is needed and the protocol terminates at the end of stage 1.

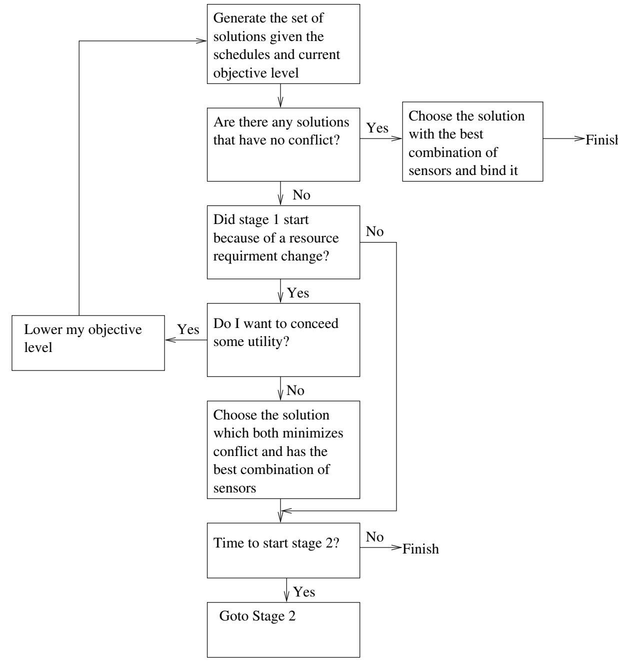
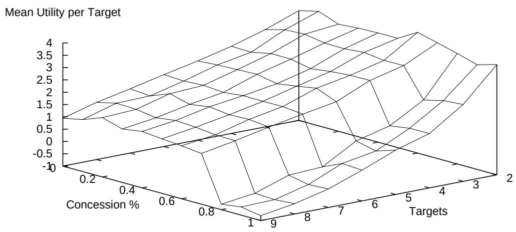


Fig. 2. Stage 1 of the SPAM protocol.

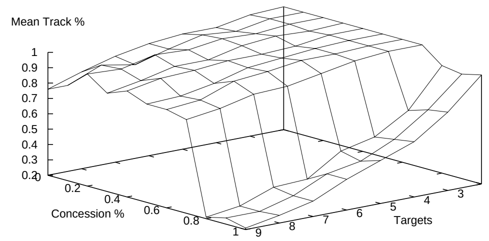
We should mention that a trade-off exists between communication overhead/solution speed and utility based on the selection of the objective level that was set at the startup of the protocol. If each of the managers chooses to use every available resource (sensors able to see their target), the possibility 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.

To take advantage of this trade-off, stage 1 was designed with a feature, called *utility concessioning*. The key idea behind utility concessioning is that often, small changes in a manager's local utility can both remove all of the conflicts on its resource assignment (thus improving global utility) and prevent it from having to wait for a mediation session to finish. In SPAM, we have a parameter, called the concession rate, which 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. 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 mediate more frequently preventing unnecessary loss of utility.

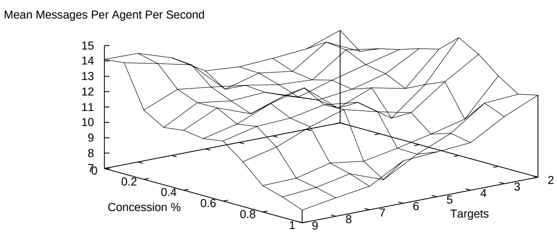
To demonstrate the effects of the concession rate on the SPAM protocol, we conducted a series of tests that varied the concession rate and the number of targets within a fixed 20 sensor environment. So, as the number of targets increases, the resource contention over the sensors increases as well. Each data point represents the average over 50 runs where both the targets and sensors are placed in the environment at a fixed, random location then the SPAM protocol is run until the agents reach a solution. A total of 4400 test runs were conducted to collect this data.



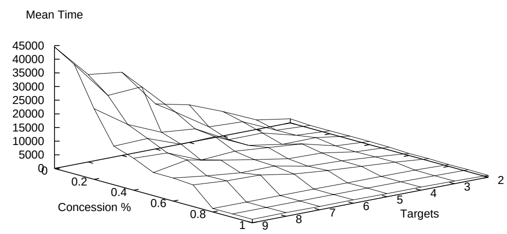
(a) Utility



(b) Tracks



(c) Messages



(d) Convergence Time

Fig. 3. Results of experimentation with the utility concessioning used in Stage 1 of the SPAM protocol.

The results of these experiments can be seen in figure 3. As you can see from the graphs, the local concession rate has a profound effect on the overall systems utility, the number of targets being tracked, the convergence time, and the number of messages. For most scenarios a high concession rate leads to relatively high utility while saving vast amounts of communication and computation. For example, for 9 targets, graph 3(a) shows that the utility is not dramatically effected by increasing the concession rate, but according to graph 3(d), increasing the rate considerably improves the convergence time for the protocol. The effects of loosing local utility become apparent at higher concession rates though. The dramatic drop-off that occurs at a rate of 0.8 is caused by agents conceding all of their local utility in order to become conflict free. Essentially, the agents begin to ignore their targets by conceding all of their local utility in order to avoid having to mediate.

The second function of stage 1 is to ensure that some degree of utility is obtained as soon as possible whenever the protocol is started due to a resource requirement change. This solution, although not conflict free, has the ability to obtain utility while the manager tries to get a better solution by going into stage 2. Conflicts that are unresolved during this period of time are 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 order to solve such conflicts. To the track manager, whether or not they get a measurement from a conflicted sensors is probabilistically random.

The third function of stage 1 is to provide the protocol with anytime characteristics. Because a solution is always derived and applied during stage 1, managers don't necessarily have to enter stage 2. They can stop the process at the end of stage 1 and accept the results that they have achieved. This is often done if a target's movement causes the resource needs to change faster than the expected time it would take to complete stage 2. The expected time to complete stage 2 is computed based on both previous experience and the current estimated communication speeds for the track managers that would be in the mediation session.

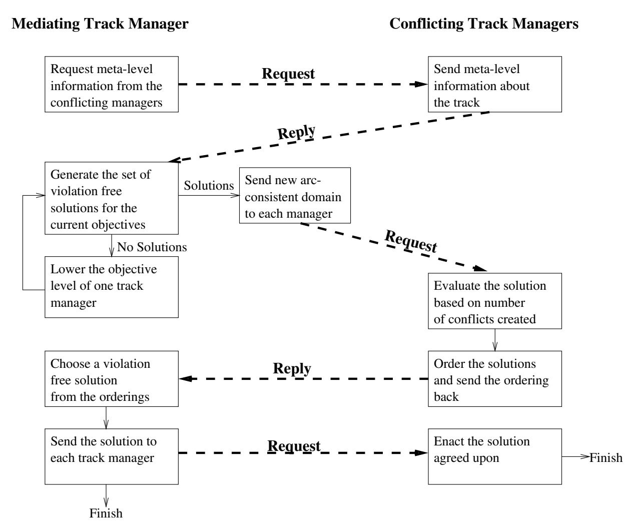


Fig. 4. Stage 2 of the SPAM protocol.

B. Stage 2

Stage 2 is the heart of the SPAM protocol (See figure 4). Stage 2 attempts to resolve resource contention by elevating the discussion to the track managers that are in direct conflict. To do this, one of the track managers takes the role of the mediator for the local conflict (note that multiple mediation sessions 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 these solutions are generated without full global information, however, the final solution may lead to newly introduced non-local conflict. If this occurs, another track managers can choose to take over the role of mediator in order to correct these newly introduced conflicts if they have the time. So, what started out as a new target or resource requirement change, may lead to a number of mediation sessions propagating across the problem landscape.

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. You should also note that the set T_m may not strictly include every one of the mediator's neighbors. Some track managers may not be using be a resource from $R(t_m)$ even though that resource belongs to their $R(t_i)$ and therefore cannot be seen by the mediator (i.e. the mediator is unaware of their relationship).

The best way to explain how stage 2 operates is through an example. Consider figure 5. 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.

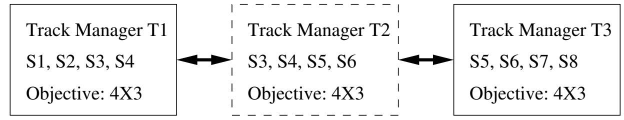


Fig. 5. Example of a common contention for resources. Track manager T2 has just been assigned a target and contention is created for sensors S3, S4, S5 and S6.

Following the protocol for the example in figure 5, track manager T2, as the originator of the conflict, takes on the role of 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. Note that sensors S1, S2, S7, and S8 are not in direct conflict and therefore are not mentioned by T1 and T3.

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 III-D). 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 4, 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 session. 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 sessions 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 [5]–[7] choose a subset of the constraints to satisfy, we actually change the structure of the constraints, removing them by lowering the objective levels, until the problem becomes satisfiable. We also differ from the Distributed Constraint Optimization (DCOP) [12], [13] work in that although DCOPs have a utility function over the possible assignments to a problem, methods for solving them do not change the underlying CSP to ensure satisfiability.

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 session 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 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 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.

The purpose of this message is actually two-fold. The first purpose is to obtain information about the effect of imposing a particular solution. The second purpose is to obtain a lock from the conflicting manager. This lock prevents the manager from changing its value while it is in a session which allows multiple sessions to occur simultaneously in the environment. If the manager is already locked, it informs the mediator who simply drops them from the session. This, of course, means that the overall session may not end with an entirely conflict-free solution, but in most cases allows the mediator to correct some of the conflicts while it waits for the lock to clear.

Each of the managers that remains in the session, using its set d_i and a revised objective level, determines 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. T1 is only sent 24 alternatives because, only 24 of its elements from the set $D(t_1)$ exist in the set $\mathbf{A}$. This means that most of the elements from $D(t_1)$ do not appear in d_1 because they were not consistent with at least one combination of elements from $D(t_2)$ and $D(t_3)$.

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 [14] 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.

	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	

Fig. 6. A solution derived by SPAM to the problem in figure 5. The table on the left is before track manager T2 mediates with T1 and T3. Notice that a number of slots have two or more tasks scheduled. The table on the right is the result of stage 2 which is conflict-free.

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 [11]; 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 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 $(t_h, t_{h-1}, \cdots, t_0)$. 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 if it chooses.

Figure 6 shows the starting and ending state of the resource schedules for the example problem. The columns represent the slots within the periodic schedules of the sensors. The rows represent the sensors. Notice that before T2 mediates, sensor S4 has two managers, T1 and T2, scheduled during every slot. After the mediation ends, all of the conflict has been removed and each manager obtains a 3×2 configuration with T1 alternating the use of S3 and S4 in slot 2 and 3.

C. Oscillation

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 mediation with track manager T2. In addition assume that T2 had previously resolved a conflict with manager T3, that terminated with neither T2 or T3 having unresolved conflict. Now when T1 mediates 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, 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 [11], [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 mediated 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 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 [3], 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 session to session and rely on the environment to break oscillations through the movement of the targets, asynchrony of the communications, timeouts, etc.

D. Generating Solutions

Generating the set $\mathbf{A}$ 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 $\forall_a\ a \in d_i \rightarrow a \in R_m$. 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 to it. 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.

This 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 $\mathbf{A}$. If there are no branches that meet this criteria, then the problem is considered over-constrained.

E. Handling Dynamics

By far one of the most interesting characteristics of the SPAM protocol is its ability to operate in environments that are highly dynamic. The SPAM protocol employs a number of techniques to deal with the effects of environmental dynamics both from a global perspective and a local perspective. One of the most useful techniques that SPAM employs is the localization of mediation sessions. By limiting the context of the problem solving to only its immediate neighbors, agents can rapidly generate solutions to considerably smaller problems than would be faced by centralizing the entire problem, computing a solution, and later redistributing an answer. This technique alone would be no better than a one look-ahead greedy method however, if it weren't for the use of overlapping context in the problem solving and the ability for managers to propagate the conflicts. Globally, this leads to a great deal of parallelism in the search although it may lead to suboptimal solutions.

Within an individual session, SPAM handles dynamics by having both a multi-stage and multi-step mediation process. By breaking apart the protocol into 2 stages, SPAM can stop processing after stage 1 if it either predicts that it will or actually does run out of time during stage 2. In addition, within stage 1, an agent can concede some of its local utility in order to avoid engaging in time consuming mediation sessions and try to find solutions that only require localized changes to the resource schedules.

The mediation session itself is broken into several distinct phases. Mediators can place deadlines on each of these phases and at any time can drop another agent for the session or terminate it all together. Although not currently implemented, it is easy to see that a scheduler could be used to set these deadlines based on the expected duration of a resource need, the expected communications delay with individual agents, etc. In fact, mediators can even place deadlines on their internal searches. The algorithm used by the agents to generate solutions can be terminated at any time and will return the set of the solutions generated up to that point.

Lastly, the mediation itself is limited to the sensors that the mediator wishes to use. That means that track managers within the session are only given schedules for the sensors that are desired by the mediator and have considerable flexibility in the actual implementation of their local solutions. For example, let's say that a mediator T1 concludes a session with another manager, T2 which involves a single sensor S1. The solution T1 has generated has T2 only using S1 during the third slot of its schedule. T2 is free to implement any local solution, as long as it doesn't use S1 during its first or second slot. In fact, if T2's target moves outside of the view of S1 during the session, it can decide not to use S1 at all.

IV. TESTING

SPAM was implemented and successfully tested in the environment described in section II. However, do to the variability created by using actual hardware, properly testing SPAM was problematic. Thus, to more systematically and rigorously evaluate the SPAM protocol, we implemented a model of the domain in a simulation environment called Farm [16]. 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 for testing SPAM has both sensor and track manager agents. Each of the sensor agents represents a single sensor which was placed in a fixed location within 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 can 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.

We ran two test series, one to test the effectiveness of our approach and the other to test its scalability.

A. *Effectiveness*

For the first test series, we wanted to determine the effectiveness and runtime characteristics of the protocol given different levels of resource contention. In this test series, we randomly placed 20 sensors within the environment and between 2 and 9 concurrent targets. Each of the targets maintained a static location throughout the run to allow the protocol to reach quiescence for the sake of measuring the convergence time.

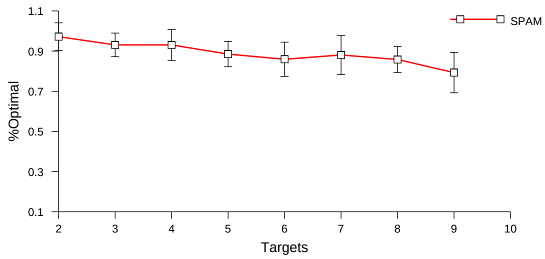
For comparison purposes, we also implemented functions to compute solutions that:

1) Would be obtained by greedy agents.
2) Have the optimal utility.
3) Track the optimal number of targets.

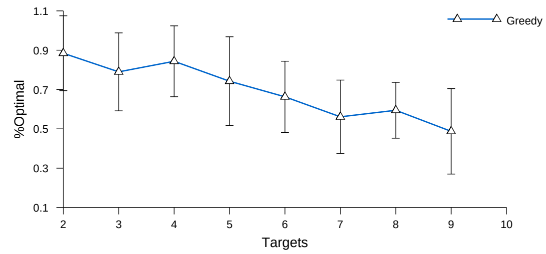
Greedy agents each request all of the available (can see their target) resources to track their targets. These requests may potentially overriding each other in the sensors' schedules leading to poor performance in areas of high contention.

The optimal utility algorithm computes the maximal set of objective levels that is satisfiable in the environment. This is done by having the algorithm 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 III-D. To make the search go faster, we prevent it from checking satisfiability on solutions that have utilities less than the best already obtained (i.e. Branch and Bound [5]), and do a simple arc-consistency check (using the pigeon hole principle) to prune obviously over-constrained problems.

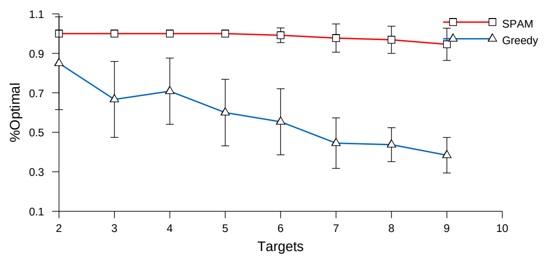
The algorithm used for finding the optimal number of tracks determines the largest number of targets that can be tracked given the available resources. For clarity, a target is considered tracked if one coordinated triangulation occurs from three or more sensors during a given period. To obtain the optimal number of tracks, a search similar to the optimal utility is done. In this search, the only objective levels that need to be



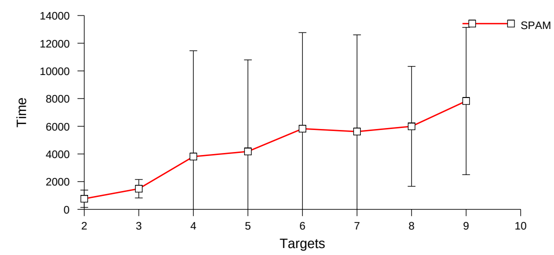
(a) SPAM Utility



(b) Greedy Utility



(c) Tracks



(d) Time

Fig. 7. Results of 20 sensor and varying target experiments comparing Greedy, SPAM and optimal allocations.

checked are either a minimal tracking (i.e. 3×1) or no tracking at all (0×0) making this search very fast.

We compared greedy and SPAM based on their achieved utility and the number of targets they tracked as a percentage of the optimal values over 20 test runs. A total of 180 tests were conducted for this series.

Figures 7(a), 7(b), 7(c), and 7(d) summarize the results of this series. As you can see from the graphs, 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. 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. Figure 7(d) 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. It should also be noted that the optimal solution took between a few seconds (for two targets) to several days (for nine targets) to compute.

Something we were not able to show in the graphs is that there are cases when the greedy algorithm 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.

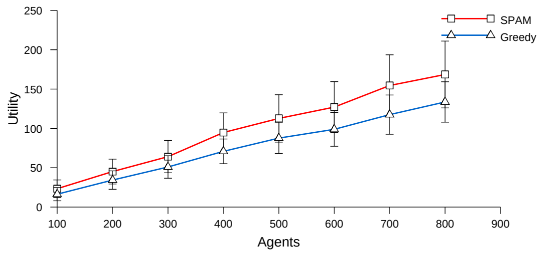
Lastly, there was at least one case where SPAM entered an oscillation. The utility obtained during the oscillation varied only slightly and the number of unresolved global conflicts fluctuated back and forth from 2 to 3. As mentioned earlier, this is a result of the localization of the search and in a dynamic environment probably would have been eliminated due to the targets' motion.

B. Scalability

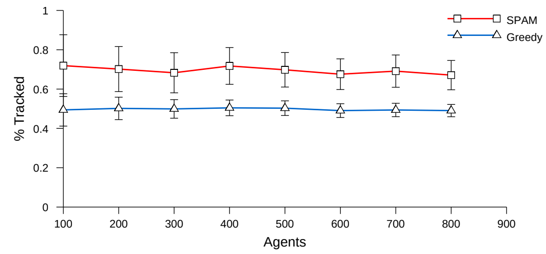
For the second simulation series we wanted to investigate the scalability of the protocol given a fixed level of contention and fixed sensor field density. In these experiments a fixed ratio of 2.5 sensors per target were used while varying the number of agents, n, from 100 to 800. This ratio was chosen because it represents a fairly over-constrained problem since each track manager needs three sensors to track its target. The field density was fixed at 4 sensors per point which ensured overlap of the resources desired by the agents. The width and height of the environment were calculated as follows:

$$width = \sqrt{\frac{s\pi r^2}{4}}$$

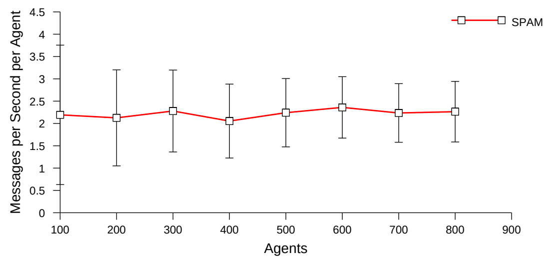
where s is the number of sensors and r is the sensors viewable radius (20 feet for these sensors). So, for 700 agents we would have 500 sensors in an environment of $396ft \times 396ft$ with 200 targets which all move with a uniformly random speed between 0 and 2 feet per second. Each of the 20 simulation runs lasted three minutes and were on a different sensor field layout. So, the values reported here are an average



(a) Utility



(b) Tracks



(c) Messages

Fig. 8. Results of scale experiments conducted with a field density of 4 sensors per point and resource ratio of 2.5 sensors per target.

over 1 hour of runtime. For comparison purposes, we also ran the greedy algorithm once again.

One very important thing to note is that the greedy algorithm is part of the simulation and therefore is given access to the global state and is not penalized for computation time or communication delays. This means that it computes a solution to a static problem at each instant of time. The targets are stopped while it computes to ensure that the problem state does not change before it determines its answer. Overall, this means that the results returned by the greedy algorithm over-estimates the utility that greedy *agents* would obtain.

SPAM, on the other hand, must explicitly communicate to gain information, is explicitly charged for computation time, works with incomplete and inaccurate information due to the targets continuous motion, as is not given credit for its solution until it is actually implemented in the sensor agents. Overall, the utility values calculated for SPAM are a very accurate representation of the actual values that would be obtained in real-time environments.

Figures 8(a), 8(b), 8(c) show the results for this series. As can be seen, as the number of agents increases linearly, so does the the utility for SPAM and the greedy algorithm, which is not entirely surprising. Notice though, that even with the large advantage that the greedy algorithm is given, SPAM consistently outperforms it.

The two other interesting results from these experiments are the percentage of targets tracked and the number of messages being used by the agents. As the number of targets increase, the percentage of targets being effectively tracked remains almost constant and the number of messages being communicated by each agent per second remains constant as well. This would suggest that the methods being used by SPAM to break apart the multi-linking of interdependencies between the track manager agents is actually very effective. Independent analysis of the SPAM protocol presented in [17] verifies these findings.

V. CONCLUSION

In this article, we described a distributed, cooperative mediation-based 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, applying arc-consistency techniques to quickly prune the search space, breaking the protocol into multiple stages and phases to allow it to make time/quality trade-offs appropriate for current conditions, and minimizing the effects of long chains of interdependencies by localizing the scope of individual mediations.

As it turn out, the core ideas used in SPAM, particularly cooperative mediation, work quite well for solving static distributed problems, including distributed constraint satisfaction (DCSP) and distributed constraint optimization (DCOP). Our current work has focused on exploiting the power of this general technique for solving problems within these areas. As such, we have developed a complete algorithm, called *asynchronous partial overlay* (APO) [18], for DCSPs and an optimal algorithm, called *optimal asynchronous partial overlay* (OptAPO) [19], for DCOPs that are based on the concept of cooperative mediation. These algorithm are, to the best of our knowledge, the fastest known methods for solving problems of these types.

Unfortunately, even though these algorithms are the fastest known, they still cannot operate in dynamic environments as they are unable to cope with rapidly changing conditions. This fact necessitates the existence of algorithms and techniques that perform both good enough and fast enough, like SPAM. The results of this work are encouraging, and although we consider the problems associated distributed resource allocation in dynamic environments to be an open research question, we feel that SPAM is a step in the right direction.

ACKNOWLEDGMENTS

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.

The authors would like to thank Tim Middlekoop, Jiaying Shen, and Regis Vincent for helping during the initial protocol development. We would also like to thank Bryan Horling for developing the Farm simulation environment used extensively for testing.

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*, vol. 21, no. 6, Nov. 1991.

[2] R. G. Smith, “The contract net protocol: High-level communication and control in a distributed problem solver.” *IEEE Transactions on Computers*, vol. 29, no. 12, pp. 1104–1113, 1980.

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

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

[5] E. C. Freuder and R. J. Wallace, “Partial constraint satisfaction,” *Artificial Intelligence*, vol. 58, no. 1–3, pp. 21–70, 1992.

[6] K. Hirayama and M. Yokoo, “Distributed partial constraint satisfaction problem.” in *Principles and Practice of Constraint Programming (CP-97)*, ser. Lecture Notes in Computer Science, G. Smolka, Ed., vol. 1330. Springer-Verlag, 1997, pp. 222–236.

[7] ——, “An approach to overconstrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction,” in *International Conference on Multi-Agent Systems (ICMAS)*, 2000.

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

[9] 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 5th International Conference on Autonomous Agents*. Montreal: ACM Press, June 2001, pp. 417–424. [Online]. Available: http://mas.cs.umass.edu/paper/199

[10] R. Vincent, B. Horling, V. Lesser, and T. Wagner, “Implementing Soft Real-Time Agent Control,” in *Proceedings of the 5th International Conference on Autonomous Agents*. Montreal: ACM Press, June 2001, pp. 355–362. [Online]. Available: http://mas.cs.umass.edu/paper/198

[11] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, “The distributed constraint satisfaction problem: Formalization and algorithms,” *Knowledge and Data Engineering*, vol. 10, no. 5, pp. 673–685, 1998.

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

[13] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, “An asynchronous complete method for distributed constraint optimization,” in *Proceedings of the Second International Joint Conference on Autonomous Agent and Multiagent Systems (AAMAS-03)*, Melbourne, Australia, July 2003.

[14] 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*, vol. 58, no. 1-3, pp. 161–205, 1992.

[15] 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*, 1992, pp. 614–621.

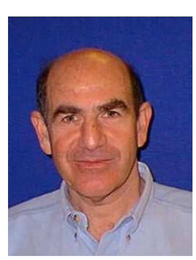
[16] B. Horling, R. Mailler, and V. Lesser, “Farm: A Scalable Environment for Multi-Agent Development and Evaluation,” *Proceedings of the 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003)*, pp. 171–177, May 2003. [Online]. Available: http://mas.cs.umass.edu/paper/243

[17] G. Wang, W. Zhang, R. Mailler, and V. Lesser, *Analysis of Negotiation Protocols by Distributed Search*. Kluwer Academic Publishers, 2003, pp. 339–361. [Online]. Available: http://mas.cs.umass.edu/paper/249

[18] R. Mailler and V. Lesser, “A Mediation Based Protocol for Distributed Constraint Satisfaction,” *The Fourth International Workshop on Distributed Constraint Reasoning*, pp. 49–58, August 2003. [Online]. Available: http://mas.cs.umass.edu/paper/250

[19] ——, “Solving Distributed Constraint Optimization Problems Using Cooperative Mediation,” *Submitted to AAMAS 2004*, 2004. [Online]. Available: http://mas.cs.umass.edu/paper/355

Roger Mailler is a PhD candidate at the University of Massachusetts Multi-Agent Systems (MAS) Lab. He received a BS with Honors in computer science from the State University of New York at Stony Brook in 1999. His main research interests are distributed problem solving, multi-agent systems organizational design, and machine learning.



Victor Lesser received his Ph.D. from Stanford University in 1972 and has been a professor of computer science at the University of Massachusetts at Amherst since 1977. He is a founding fellow of AAAI, and the founding president of the International Foundation for Multi-Agent Systems. His major research focus is on the control and organization of complex AI systems. He has been working in the field of Multi-Agent Systems for over 25 years. Prior to coming to the University of Massachusetts, he was a research scientist at Carnegie-Mellon University where he was the systems architect for the Hearsay-II speech understanding system, which was the first blackboard system developed.