Soft-Real Time, Cooperative Negotiation for Distributed Resource Allocation

Roger Mailler, Regis Vincent, Victor Lesser
Jiaying Shen
University of Massachusetts
Department of Computer Science
Amherst, MA 01003
{mailler, vincent, lesser, jyshen}@cs.umass.edu

Tim Middlekoop
University of Massachusetts
Department of Mechanical and
Industrial Engineering
Amherst, MA 01003
mtim@farm.ecs.umass.edu

Abstract

In this paper we describe an approach to cooperative negotiation that uses a combination of techniques to allow our distributed resource allocation negotiation protocol to conform to soft real time constraints while obtaining reasonable solution quality. Amongst these techniques are the ability to resolve conflict in the allocation of resources on multiple levels, temporarily binding and incrementally improving the quality of the solution (a form of distributed hill climbing) given time constraints providing for an anytime characteristic, and restricting the context of negotiations to only use local information with extended meta-level information to generate and propose possible solutions to the problem. We describe the implementation of a simulator for the protocol and experimental results.

Keywords: Multi-Agent Systems, Negotiation

Introduction

Resource allocation is a classical problem that has been studied for years by Multi-agent Systems researchers (Smith 1980). The reasons for this is that resource allocation is difficult and time consuming to do in a centralized manner when the environment is dynamic and the time or cost of centralizing the information needed to generate a solution is considerable. Negotiation, a form of distributed search (Moehlman, Lesser, & Buteau 1992), has been viewed as a viable alternative to handling this complex search through a solution space that includes multi-linked interacting subproblems.(Conry *et al.* 1991) Researchers in this domain have focused primarily on resource allocation problems formulated as distributed constraint satisfaction problems(Yokoo 1998). In this work, we extend this classic formulation in two ways. First, we introduce dynamic soft-real time constraints which requires the negotiation protocol to adapt to the available time left. This estimation is determined dynamically as a result of emerging environmental conditions. Second, we reformulate the resource allocation problem as an optimization problem in which there are a range of acceptable solutions with varying preferences.

In this paper, we present a negotiation protocol that exploits the fact that the agents within the system are cooperative and have the ability to resolve conflicts internally. This means that conflict can be left unresolved as a result of negotiation, but the agent faced with the conflict must resolve it based on its local perspective. We are not making the assumption that these internally based solutions obtain the best possible results, but that they are capable of providing some measure of utility while a better solution is obtained. The ability to create temporary solutions and incrementally improve them both locally and globally forms a distributed hill climbing search through the solution space that optimizes based on the demands of a soft real-time environment. In this context, soft real time should be interpreted as soft deadline, which means that finishing a task a bit early or late does not result in detrimental effects.

Our negotiation protocol is based on three major principles which allow it to accomplish the previously mentioned goals. First, we limit the context of the negotiation such that allocation problems are always resolved locally with only limited information about interacting subproblems being considered. After local negotiation is finished, each of the agents can choose to propagate the negotiation in an attempt to resolve conflict that may have been created as a result of the originating negotiation. Viewing this activity from the perspective of the global problem, each of the agents that propagates the negotiation is in essence locally optimizing in an attempt to reach a global optimum which is form of distributed hill climbing. Second, local negotiations are conducted at multiple levels of abstraction. Agents can choose to resolve the conflict at different granularities and if they are unable to resolve it at one level, because of limited time, can leave it to be handled at lower levels. Lastly, we have designed the negotiation protocol to exploit the previously mentioned techniques to have an anytime flavor. By having the ability to take solutions that may have unresolved conflict and still obtain some quality, we can actually bind temporary solutions while attempting to resolve the conflicts at a lower level. Clearly in a dynamic domain this has the ability to "buy some time" so that a good solution can be obtained while not completely abandoning processing that has some value in the mean time.

In the remaining sections of this paper, we introduce a distributed monitoring and tracking application which motivated the development of our protocol. Next, we describe the Scalable Protocol for Anytime Multi-level negotiation(SPAM). In the results section of the paper, we will describe an abstract model of the task environment that was used to develop and test SPAM in addition to some of the

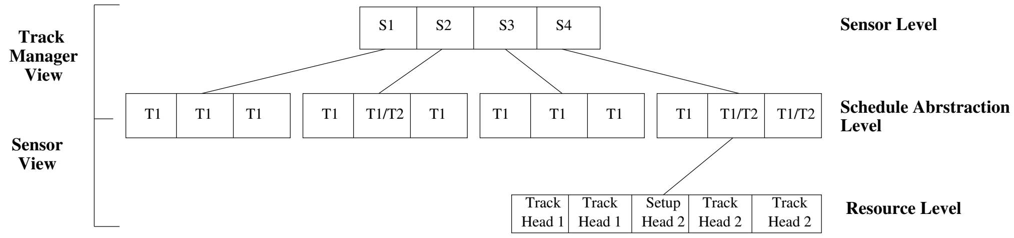

Figure 1: *The three level of abstraction used to negotiate when tracking a target. The top level shows track manager T1's sensor level abstraction wanting to use sensors S1, S2, S3, and S4. All conflict could not be resolved at a higher level so sensors S2 and S4 are left to resolve the conflict between T1 and T2 at the next lower level. Sensor S4 chooses to resolve its conflict at the resource level by squeezing in two track tasks from T1 followed by two from T2.*

early results we have obtained. The last section of the paper will present conclusions and future directions for this work.

Domain

The problem that we are exploring is that of allocating sensor time to the task of tracking targets. In this problem, multiple sensors platforms are distributed with varying orientations throughout a real time environment (Horling *et al.* 2001). These platforms have three radar based sensors each with a viewable 120 degree 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 a coordinated measurement of the target which is then fused to triangulate the target's position. Having more sensor heads, taking measurements more often, or having tighter relative synchronization of the measurements yields better overall quality in estimating the targets location and a more optimal result. The sensor platforms are restricted to only taking measurements from one sensor head at a time which is the key restriction forming the basis of the resource allocation problem.

Each of the sensor platforms has some limited computational ability and in our system is populated 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 maintain a high degree of local autonomy, being able to make trade-off decisions about competing tasks using our Soft Real Time Architecture (SRTA pronounced Serta)(Vincent *et al.* 2001).

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 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 act as the focal point of negotiation that take place as part of solving any resource contention that may arise while tracking the target.

To lend to the real time characteristics of this problem, targets continuously moves through the environment as a scenario unfolds. This means that during the course of a run, targets move from the viewable range of some sensors to others. This, of course, means the actual allocation problem changes in structure during the course of a run as the track managers alter their resource requirements due to the discovery of new targets and the movement of existing ones.

Contention is introduced when more than one target enters the viewable range of a single sensor platform. Because of the time it takes to perform a measurement and the one measurement at a time restriction, track managers have to come to some sort of agreement about how to split the resource while still being able to track their target. This local agreement can have profound global implications, however, what if as part of the local agreement one track manager completely relinquished control of a sensor platform and takes another instead? This may introduce contention with another track manager which could propagate through the entire environment.

Abstraction

The actual resource allocation problem that is created by this environment can be view at different levels of abstraction (see figure 1). At the highest level is the sensor level. This level is maintained by the individual track managers and strictly focuses on which sensors are needed and desired to track the target. Solutions created at this level ignore the details of the individual sensors' schedules in making choices of how to allocate resources and simply choose based on the track managers internal requirements. Of course, since these solutions are created without information about what the sensors are actually doing, they are almost never free of conflict.

The next level of abstraction is the schedule abstract level. At this level, tasks can be viewed as periodic (which tracking is) and resource schedules can be viewed at a course slot based granularity (all measurements take approximately the same amount of time). Within the sensors platforms, our agents maintain the schedule abstraction level by using the Periodic Task Controller(PTC). The PTC is a slot based, periodic scheduler that feeds SRTA with tasks at times that are appropriate to its schedule. The PTC is capable of resolving conflict by using one of several techniques including shifting slot boundaries, selecting tasks to execute based on importance level, or temporarily shifting a task to empty slots in its schedule. It is easy to see that if a negotiation ends in unresolved conflict, which we call a co-binding, that the PTC has some capability to resolve the conflict. For instance, if two track managers T1 and T2 are in conflict over sensors

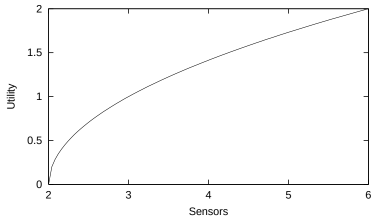

Figure 2: *Utility of taking a single measurement from T_a sensors.*

S4, they may, due to time constraints, be unable to resolve the issue and may leave one of the slots of S4 co-bound. This means that when the PTC in S4 attempts to schedule that slot, it is forced to make the local determination as to which of the track managers gets the slot for that period.

At the lowest level, the resource level, all of the minute details of task execution and resource usage within the sensors is scheduled using SRTA. If scheduling conflicts reach this level of abstraction the Partial Order Scheduler(POS), a component of SRTA, can shift the task execution to try to eliminate any remaining conflict. Conflicts at this level can be created because the sensor is working on a non-tracking task that is not explicitly reasoned about at the schedule abstraction level.

During the course of negotiation, due to time constraints, the track manager can choose to operate at either the sensor or schedule abstraction level of negotiation. Leaving unresolved conflict at these levels of abstraction, though, introduces a great deal of uncertainty about the exact nature of the final solution. The deeper the track manager is able to go and resolve conflict, the greater the guarantee about the solution quality obtained in the end.

Utility

To help clarify what our protocol is attempting to achieve it helps to see how utility is measured in the tracking domain. As mentioned previously, 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 2. Increasing the measurements taken in a given period of time yields a linear increase in the overall quality of the track.

If we say that T_a is the number of sensors that took measurements leading to the positional estimate and T_s is the number of times they are taken in a given period of the abstract periodic schedule, then we can quantify this relationship by the following formula:

$$Util(Track) = Util(T_a) \times T_s$$

In fact, track managers within the system use this measurement as the basis for the objective levels assigned to a track. We will often denote the objective level as $D_a \times D_s$ denoting the number for agents desired for the number of slots in the schedule abstraction level. For example, a track manager may wish to have three agents for two slots of the schedule abstraction level denoted 3×2. For this domain, we typically set the number of slots at the schedule abstraction level to match the number of sensor heads on each platform which is three.

Information	None	Local Only	Local with Meta-level
	Stage 0	**Stage 1**	**Stage 2**
Level	Sensor	Schedule Abstraction	

Figure 3: *The three stages of SPAM showing the information that is available and the level of abstraction the track manager uses in generating possible solutions.*

Looking at this utility function it should be noted that co-binding can have a profound effect on the quality of a track. In fact, because the sensors make the decision about which track to satisfy on each period of their periodic schedule, having more than one sensor bound for a particular slot causes a near random occurrence of synchronization. For example, if a track manager T1 uses sensors S1, S2, S3, and S4 each for one slot of their schedule and sensors S2 and S4 are co-bound on that slot with one other track manager, it is easy to see that T1 has a 25 percent chance of getting four sensors for the slot and a 50 percent chance of getting three sensors for that slot during any given period. This relationship can be seen in the following formula. Here S is the set of slots in the abstract schedule level and T_i^s is the number of actual measurements that are taken during a given slot s .

$$Util(Track) = \sum_{s \in S} \sum_{a=3}^{\infty} Prob(T_i^s = a) Util(T_i^s)$$

Finally, the global utility can be calculate from the following formula which just says that the overall utility is the sum of the utilities for the individual tracks (one track per target).

$$Utility = \sum_{Track \in Targets} Util(Track)$$

Protocol

To meet the objectives of the environment and to incorporate the techniques that were discussed in the previous sections, the Scalable Protocol for Anytime Multi-level (SPAM) negotiation is divided into three stages. As the protocol transitions from stage to stage, the agent acting as the track manager gains more context information and therefore is able to improve the quality of its overall decision. After each stage or at anytime during stage 2, 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). Figure 3 shows the amount of information that the track manager has at each stage of the protocol. The figure shows that as the amount of information obtained increases, the track manager is able to shift its negotiation abstraction level. This means that if the track manager chooses to terminate the protocol before stage 1, it acts at the sensor level of abstraction (deciding on only which sensors it desires) and leaves the decision of how to handle the actual scheduling to the sensors themselves as was discussed in the previous sections.

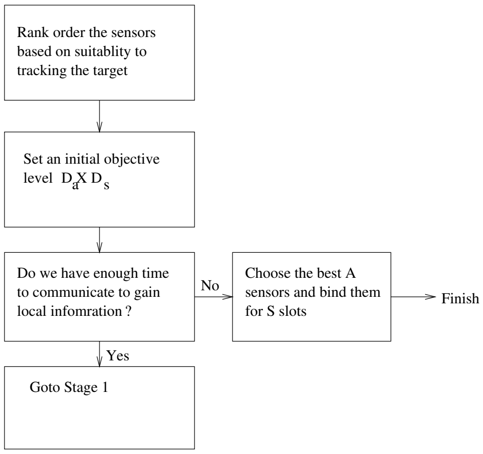

Figure 4: *Stage 0 of the SPAM negotiation protocol. Terminating at this stage forces the sensors to resolve unhandled conflicts leading to potential lower solution quality.*

Stage 0

On target detection, stage 0 (see figure 4) of the negotiation protocol is activated. Stage 0 is primarily responsible for viewing the problem at the sensor level of abstraction. Because of this, each of the sensors that have the potential to track the target are evaluated and ordered. In this stage, the track manager also assigns an initial objective level to the track. Objective levels in general are derived from the track managers objective function. This function, which may be different for every track manager, defines the order of the objective levels, the initial objective level for a track, and a lower bound of the objective level before giving up on an unconflicted solution. Changing these parameters can alter the characteristics of the search process to make it faster (start at a lower objective level) or better (start at the best possible objective level).

The actual activity of choosing a solution at this level of abstraction is primarily domain specific. For example in the tracking domain, criteria for solution choice might be the relative proximity of the target to the sensor, whether the target is moving toward or away from the sensor, etc. The solution choice, however, is based on internal information only.

As you can see from figure 4 the protocol maintains its anytime characteristic by determining whether enough time exists to go on to the next stage and down to the next level of abstraction. If time is limited then the negotiation session is terminated leaving it at a high level of abstraction. Any resulting conflict is left to be resolved by the agents that reside in the sensor platforms.

If enough time is available, the track manager can transition to stage 1 of the protocol. It should be noted that although we don't bind a temporary solution at the sensor level of abstraction, it doesn't mean we couldn't. By gaining some abstract schedule information from the sensor platforms in the beginning of stage 1, we are able to bind a considerably better solution temporarily and avoid wasting a set of bind messages to the sensor platforms.

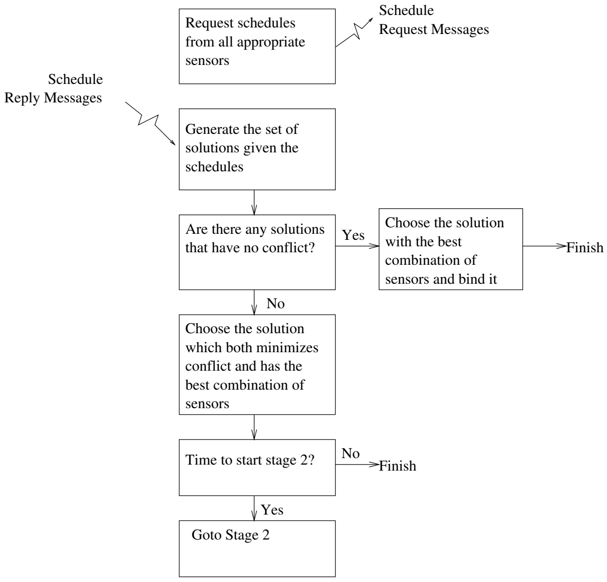

Figure 5: *Stage 1 of the SPAM negotiation protocol attempts to resolve conflict locally at the schedule abstraction level.*

Stage 1

Stage 1 of the SPAM protocol begins by obtaining abstract schedule information from the PTC in each of the sensor agents. This information is used in two ways. First, if a solution at the current objective level can be obtained, the track manager can bind the solution and avoid a more costly track manager-to-track manager negotiation process. We discuss how possible solutions are generated in a later section. Second, if a solution cannot be found at the current objective level, the track manager has enough information to bind a good solution which minimizes the amount of unresolved conflict and maximized the track manager's local objective level. Like stage 0, the negotiation session can be terminated at the end of stage 1 if enough time is not available to continue.

Solutions in stage 1 are only considered at the original objective level set forth in stage 0. The reason for this is that if the track manager were to lower its objective function without considering additional information then in all likelihood it would end up with a utility that was lower than it should have been. For example, consider the following scenario. Track manager T2 is assigned the role of tracking a target M2. T2 determines during stage 0 that it wishes to have sensors S1, S2, S3, and S4 to track the target and assigns an initial objective level of 4×3. After obtaining the abstract schedule of all four sensors the track manager finds that this solution is not possible because manager T1 has all three slots of sensor S4 assigned. Now, as the protocol stands now, T2 would bind a temporary solution and move into stage 2 to begin negotiation with T1. Clearly if T2 had lowered its objective function to 3×3 a solution (S1, S2, S3) with no conflict could have been obtained without expending time by going into stage 2. From a utility perspective, say that the other track manager, T1, was actively using a 5×3 objective level in tracking its target. If T2 ac-

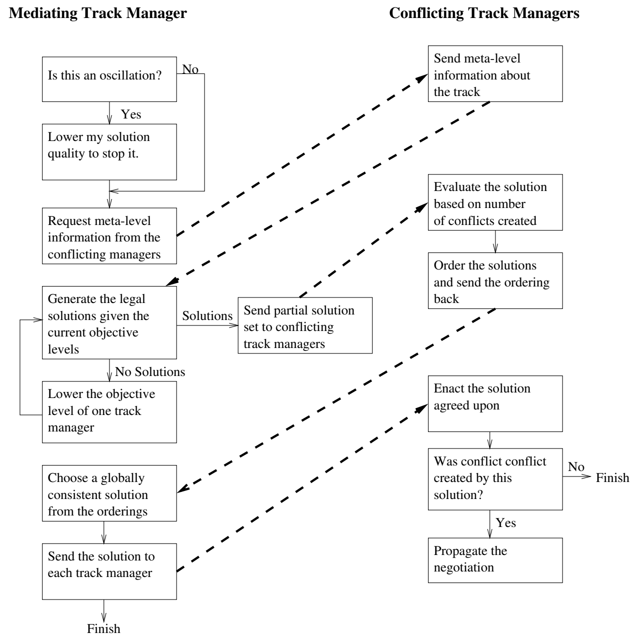

Figure 6: *Stage 2 of the SPAM negotiation protocol resolves all local conflict at the schedule abstraction level through negotiation with conflicting track managers.*

cepts a 3×3 then the global utility would be around 8.2. If, however, T2 co-binds, while negotiating, then both managers obtain a 4×3 configuration with a global utility of about 8.5. Although the difference seems minimal, our belief is that in order to maintain the hill climbing nature of the search, agents must always try to locally maximize their utility until such a time where it is determined that to do so actually harms the global utility.

Stage 2

Stage 2, the final stage of SPAM, is the heart of the negotiation protocol (See figure 6). Stage 2 attempts to resolve all local conflict 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 (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 possible 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 in order to resolve this conflict if they have the time. So, what started out as a new target or resource requirement, may lead to the negotiation propagating across the problem landscape.

Stage 2 starts in the oscillation detection phase. Oscillation occurs because, as previously mentioned, conflicts are resolved locally without regard to the global context. For example, say that track manager T1 originates a negotiation with track manager T2. In addition let's say that T2 had previously resolved a conflict with manager T3, that terminated 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 ended up in conflict 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.

To prevent this from happening, each track manager maintains a history of the sensor schedules that are being negotiated over whenever a negotiation terminates. By doing this, managers are 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 lowers its objective level to force itself to explore different areas of the solution space. It should be noted that in certain cases oscillation may be incorrectly detected using this technique which can result in having the track manager unnecessarily lower its objective level.

After the mediator concludes the oscillation detection phase, 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. Using this information, the mediator can begin to generate possible solutions to the problem. As you can see in figure 6, the mediator enters a loop that involves attempting to generate solutions followed by lowering one of the track manager's objective levels. This loop is terminated under one of two conditions. First, if given the current objective levels for each of the track mangers, a set of unconflicted solutions is available (the next section explains how solutions are generated), the negotiation enters the solution evaluation phase. Second, the objective levels of the track managers cannot be lowered any further. Under this condition, the negotiation session is terminated and the mediator takes a solution at the lowest objective level that minimizes the resulting conflict conceding that it cannot find an unconflicted solution.

During the solution evaluation phase, the mediator sends each of the track managers the set of possible alternative solutions for the sensors that are in conflict between the mediator and the track manager. The track manager, using this information and the proposed objective level, can then determine what solutions, if any, are acceptable. The evaluation phase ends in having an order assigned for the solutions from each of the track managers. The mediator can then choose what overall solution to apply by choosing the highest ranked solution from each of the managers which leads to a consistent solution. The order by which each manager gets picked is determined by the amount of additional conflict the track manager has. For example, say T1, T2, and T3 were in negotiation and T1 was acting as the mediator. When T2 and T3 return their meta-level information, it is determined that T3 has external conflict for two of its possible sensors which are not in direct conflict with T1. During the solution evaluation phase, T1 will pick a consistent solution by choosing T3's highest ranked solution, then T2's highest ranked solution that is consistent with T3's choice, and finally the solution that applies to itself. By allowing T3 first choice, it improves the possibility that T3 will not have to propagate the negotiation because it may end up with an unconflicted solution.

The next phase of the protocol is the solution implementation phase. Here, the mediator sends the results to each of the track managers, who in turn implement their solutions. In addition, the state of the sensors is stored for future oscillation detection. At this point, the track managers can determine if the solution that has been selected leads to new conflict. As it currently stands any track manager that has unresolved conflict, will propagate the negotiation. In future versions, it is our hope that utility and not conflicts will form the basis for determining when to propagate.

Generating Solutions

Generating potential solutions for the domain described earlier involves taking the limited information that was provided through communications with the conflicting track managers and assuming that the sensors which are not in direct conflict, are freely available. In addition, because the track manager that is generating potential solutions only knows about the sensors which are in direct conflict, it only creates and poses solutions for those sensors. The formula below gives the basic form for how solutions are generated for a single track manager. Here, A_s is the number of slots that is available in the schedule abstraction layer, D_s is the number of slots that are desired based on the objective level for the track manager, A_a is the number of sensors available to track the target (those that can see it), D_a is the number of sensors desired in the objective function, and finally C_a is the number of sensors under direct consideration because they are conflicting.

$$Solutions = \begin{pmatrix} A_s \\ D_s \end{pmatrix} \left(\sum_{i=max(0,D_a-A_a+C_a)}^{min(C_a,D_a)} \begin{pmatrix} C_a \\ i \end{pmatrix} \right)^{D_s}$$

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 $\times$ 2 and sensors S2 and S3 are under conflict. The generation process would create the $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ 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 generating local solutions in stage 0 and 1 of the protocol. This special case generation is actually done by simple setting $C_a = A_a$. The formula above, in this case reduces to

$$Solutions = \begin{pmatrix} A_s \\ D_s \end{pmatrix} \begin{pmatrix} A_a \\ D_a \end{pmatrix}^{D_s}$$

We can also generate solutions when there are 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 this ability, we are able to generate potential solutions for the track managers in stage 2. By ordering the track managers based on their external constraints we can generate solutions for them one at a time using the results from higher precedence track managers as constraints for lower precedence ones. For example, three track managers T1, T2, and T3 are in conflict and T1 is acting as mediator. In addition, say that T2 has two conflicts and T3 has one conflict. When the potential solutions are generated, T1 generates solutions for manager T2 first. T1 then uses the results from this as constraints on the creation of solutions for T3. The resulting solutions (now with solutions for T2 and T3) are used as constraints for generating the solutions for T1.

This process forms the basis of a search for solutions to the local conflict. You can view this as a tree based search where the top level of the tree is the set of possible solutions for the most constrained track manager. Each of the nodes at this level may or may not have a number of children which are the solutions available to the second most constrained track manager and so on. The lowest level of the tree is the set of solutions available to the mediating track manager, who never has external conflict. Only branches of the tree that have leaf nodes at the mediating track manager level are considered as complete and are passed to the other track managers to rank. If there are no solutions at the lowest level, then the problem is considered over constrained.

To evaluate the SPAM protocol, we developed a simulator that uses a model of the tracking environment described earlier in the paper (see figure 7). In this simulator, we concentrated on evaluating primarily stage 1 and 2 of the protocol. To do this, the simulation was constructed using two major pieces, the environmental simulator and the track managers themselves. The environmental simulator manages the state of the sensors, spawns the track managers, introduces new targets, and manages propagation requests from the track managers. The track managers, which are defined by a set of sensors that they desire once they are given a target, handle any incoming negotiation requests from other track managers, spawn new negotiations when assigned a target, and request to be placed on the propagation list when they have unresolved conflict.

Using the environment in figure 7, we ran every possible configuration of targets and every possible order of target introduction in order to test the convergence, communication and, utility properties of the algorithm. In this environment, that equates to 9! = 362,880 tests.

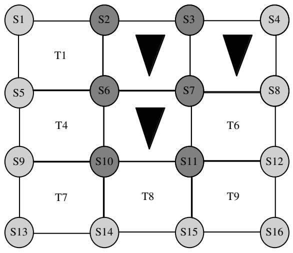

Figure 7: *The environmental layout used for testing the SPAM protocol. Track managers were defined by the sensors they needed to track targets in their region. In this figure, circles represent sensors, triangles represent targets and squares represent track manager regions.*

Results

We have done some preliminary analysis of the results which can be seen in the graphs in figure 8. These graphs show the average number of propagations and communications that occur at various degrees of difficulty (measured by the number of targets in the environment). As you can see, the SPAM protocol, when going from 0 to 9 targets, converges on a solution in an average of 18 discrete negotiation sessions which includes the 9 original local negotiations that take place due to target introduction. In addition, although not included in the graphs, on average each track manager obtains a local objective level of better than 3×1 and receives a utility of approximately 3.61. On average, the overall solution has less than 1 unresolved conflict. Communication cost is dominated by track manager to sensor communications. On average to complete a 9 target problem, it takes 163 sensor schedule requests and 155 bind messages. These numbers may appear large, but considering that the activity is being done in parallel, the bind message counts include the temporary bind messages sent out at the end of stage 1, and that schedule requests occur in several place during the protocol, these numbers seem very reasonable.

Conclusion and Future Directions

In this paper, we have described the SPAM protocol which was built to solve coordinated 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.

Much work remains to be done on this protocol. Most importantly, work needs to be done on adapting the protocol to work in other domains. In addition, we'd like to explore the protocol from a more formal perspective and try to obtain provable bounds on the utility obtained and the time to convergence under different time constraints and levels of subproblem interaction. Finally, we plan to migrate the protocol into the tracking environment described in the paper which is based on actual hardware. In this way, we can obtain empirical evidence that the protocol can meet the demands of a soft-real time, uncertain, dynamic environment.

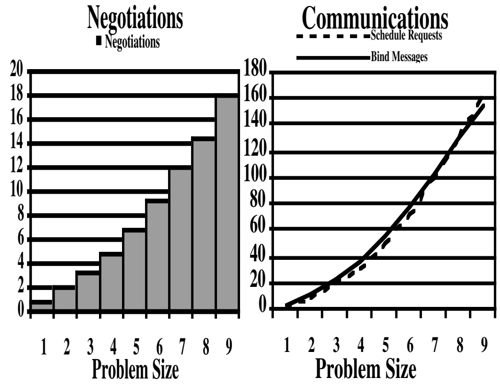

Figure 8: *Results of testing SPAM in the scenario in figure 7. These graphs represent the averages over 362,880 runs.*

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.

Thanks to Sun Microsystems for donating the Enterprise 3500 system under their Academic Equipment Award Program #EDUD-7824-000438-US, that was used to conduct the test runs of our protocol presented in this paper.

References

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

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

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

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

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

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