Incremental Negotiation and Coalition Formation for Resource-bounded Agents: Preliminary Report

Incremental Negotiation and Coalition Formation for Resource-bounded Agents
Preliminary report

Charles L. Ortiz, Jr. and **Eric Hsu** and **Marie desJardins**
Artificial Intelligence Center
SRI International
Menlo Park, CA 94025
{*ortiz* | *hsu* | *marie*}@*ai.sri.com*

Timothy Rauenbusch and **Barbara Grosz**
Division of Engineering and Applied Sciences
Harvard University
Cambridge, MA 02138
{*tim* | *grosz*}@*eecs.harvard.edu*

Osher Yadgar and **Sarit Kraus**
Department of Computer Science
Bar-Ilan University
Ramat Gan, 52900 Israel
{*yadgao* | *sarit*}@*cs.biu.ac.il*

Abstract

We explore a class of task allocation mechanisms that are incremental and can be tuned to the computational resource limitations of agents. Our focus is on distributed task and resource allocation problems involving coalitions of cooperative agents that must negotiate among themselves on the distribution of tasks. Our emphasis is on the design of mechanisms with desirable real-time and dynamic properties. We describe preliminary work in four areas: the design of what we call time-bounded commitment networks that are extensions of task-auctions and contract nets and that support a notion of reciprocal commitment; anytime algorithms for combinatorial task allocation that take into account both positive and negative task interactions, organizational frameworks for efficient task allocation in highly dynamic domains involving hundreds of agents, and logical tools for analyzing dynamic emergent properties of agent societies.

Keywords: Negotiation, multiagent systems, real-time resource allocation.

Introduction

We report on preliminary work into the design of task allocation methods that exhibit desirable real-time and dynamic properties. These methods are loosely based on two related paradigms: auctions and contract nets (Kraus 1996; Sandholm 1999; Hunsberger & Grosz 2000; Smith & Davis 1983; Sen & Durfee 1996). Auction-like mechanisms represent attractive methods for the quick and decentralized negotiation of task and resource allocations: agents need not exchange large volumes of local state information to some centralized point for an allocation. Typically, auctions take place in competitive settings: a group of agents places bids on an object that has been announced for sale and the highest bid wins. Since the focus in competitive settings is on truthfulness and fairness, mechanisms have been developed — such as second-price auctions — to ensure truthfulness. Our focus, however, is on cooperative agents; hence, truthfulness can be assumed. In the problems that are of concern to us, agents instead bid on tasks — either the cost to perform those tasks or how well they can perform them — and the best bid is assigned the task. Such task auctions are very similar to contract nets. However, the work we describe borrows ideas from combinatorial auctions in which a set of objects is announced for bid and bidders can place bids on any subset of those objects.

Within the domain of resource allocation, we are primarily interested in problem settings that are very dynamic: new tasks can appear while other tasks are executed, the processing of tasks has associated real-time execution constraints, and agent coalitions can consist of hundreds of agents. Our emphasis on solution incrementality emerged with such problem domains in mind: the time-stressed nature of such problem domains precludes the possibility of computing optimal resource allocations before execution. Instead, agents should negotiate partial, good-enough allocations that can later be refined if time permits.

The remainder of this paper summarizes work in four areas: the design of what we call time-bounded commitment networks, which are extensions of task-auctions and contract nets and which support a notion of reciprocal commitment; anytime algorithms for combinatorial task allocation that take into account both positive and negative task interactions; organizational frameworks for efficient task allocation in highly dynamic domains involving hundreds of agents; and logical tools for analyzing dynamic emergent properties of agent societies. We begin by first defining a class of distributed resource allocation problems.

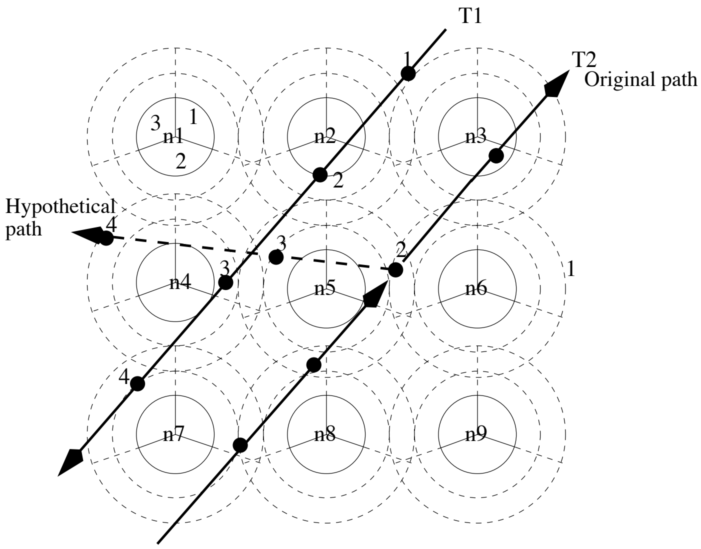

Figure 1: multisensor tracking.

Distributed resource allocation

Definition (DRAP) A distributed resource allocation problem (DRAP), D, is a triple, $\langle \mathcal{R}, \mathcal{T}, \mathcal{Q} \rangle$ where $\mathcal{R}$ represents a set of resources or agents, $\mathcal{T} \subseteq 2^T$, where T is a set of task elements, and $\mathcal{Q} : \mathcal{R} \times N \times T \rightarrow [0..1]$ associates a quality measure with each resource-time-task triple, where N is a set of integer times. The function Q is generally expressed in terms of the distance of the task segment, T, from a sensor. A cost, C, is sometimes also associated with a resource allocation: $C : \mathcal{R} \times N \rightarrow \Re$, where $\Re$ is the set of reals. A solution, s_D, to a DRAP, $D = \langle \mathcal{R}, \mathcal{T}, \mathcal{Q} \rangle$, is a mapping $s_D : T \times N \rightarrow 2^{\mathcal{R}}$.

Solutions are sometimes constrained: for example, one might want to place bounds, B_C, on the maximum group cost as well as bounds, B_Q, on the minimum quality, in the following way: $\mathcal{Q}$ such that

$$\frac{\sum_{T \in \mathcal{T}} Q(s_D(T, time(T)), time(T), T)}{|\mathcal{T}|} \geq B_Q$$

and

$$\sum_{T \in \mathcal{T}} C(s_D(T, time(T)), time(T)) \leq B_C$$

where $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ is some set of task elements and $s_D(T_i, time(T_i)) = \{R_1, R_2, \ldots, R_m\}$ is the set of resources assigned to a task segment, T_i at time $time(T_i)$.

An example of a DRAP is shown in Figure 1, involving multisensor tracking. The figure shows an array of nine doppler sensors. Each sensor has three sectors associated with it, labeled $\{1, 2, 3\}$. A sensor can turn on a sector and take both frequency and amplitude measurements to determine velocity and distance. The more sectors that are on, the greater the power usage. The farther away the target is from the sensor, the lower the quality of the measurement. At least two sensors are necessary for estimating the location of an object; three sensors are desirable for obtaining a good-quality estimate. Sectors require a 2 second warm-up time, and two objects appearing in the same sector and at the same time cannot be discriminated.[1]

In this example, we assume that the projected paths have been computed — based on initial localization, direction and velocity measurements — for each of two targets, $T1$ and $T2$. The paths are shown as dark lines; the dashed line represents a hypothetical change in direction of T2 which will be discussed later. The problem in this example is to allocate, in a distributed manner, a coalition of sensors along the paths of both targets. One way of implementing a task auction to allocate the sensors is to first assign nodes $n3$ and $n7$ each the role of auctioneer. Each would then initiate an auction for the respective tasks corresponding to an assignment of three sensors at each future time point — indicated in the figure by small dark circles — to nodes and relevant sectors. In the actual system, power consumption and communications (number of messages exchanged) are resources that must also be managed. Formally, the above example corresponds to the DRAP in which $\mathcal{R}$ stands for the sensor nodes, and $\mathcal{T}$ consists of the set $\{T1, T2\}$.

Real-time commitment networks

Commitment networks are hybrid negotiation mechanisms that borrow ideas from auctions, contract nets, and theories of collaboration. Teams negotiate the distribution of new tasks by initiating auctions to potential team members who then bid on how well they can perform those tasks. The protocols have been designed to adapt to the demands of a problem situation along a number of dimensions: for example, as in contract nets, subauctions are possible, and their depth can vary depending on the time available. However, unlike contract nets in that the commitment of a contractee to a contractor is one-sided, commitment networks allow for reciprocity between a contractee and another team member: if the contractee can identify some task that could help another team member, then it attempts to do so. Such reciprocity has proved valuable in providing a certain measure of fault tolerance. Finally, commitment networks borrow the idea of combinatorial task allocation: sets of tasks can be announced at one time and bidders can bid on any subset of those tasks.

Commitment networks introduce three new constructs not present in standard contract nets.

Persistent bids In the contract net protocol, if a bid is not accepted, it is terminated. In commitment networks, bids are allowed to persist by default. In time-critical situations, this has the advantage that if a new task appears in a location close to that covered

[1] We are using both a physical sensor suite and simulations based on the sensor suite for experimentation.

by a recent bid, the auctioneer can simply assign it to the corresponding bidder. Such an assumption is defeasible: agents can announce that the default bid is no longer valid because of some new commitment.

Reciprocal background commitments
These sorts of commitments are inspired by theories of collaboration that have argued that collaborating agents should demonstrate a willingness to provide helpful behavior to other team members (Grosz & Kraus 1996).

Contingent commitments
Dynamic worlds in which tasks can change require commitments that can be made contingent on some aspect of the future. These sorts of commitments allow an agent to drop a commitment if conditions change in a way that warrants such an action.

Our approach to the multisensor tracking problem described earlier using commitment networks involves three stages: (1) initial coalition formation, (2) formation of a future coalition based on a projected object path, and (3) refinement of an existing coalition. We use the term *coalition* to refer to a group of agents that are joining together to perform some task; the members of a coalition can change as circumstances change. The initial coalition formation is a very quick process that assigns a group of three agents to a target. The initial coalition's task is to determine the position, direction, and velocity of the target. In the second stage, one of the agents in the initial coalition takes that information and projects the path of the target into the future, and then runs an auction on some set of agents that neighbor the projected path. The projected path is represented by a cone of uncertainty: the farther into the future, the greater the uncertainty in the projected position. In the final stage, a coalition can be adapted to changes that might occur in the path of a target: when agents in a particular coalition, A, notice the change, they inform the remaining agents in the original coalition — a consequence of a reciprocal background commitment — of that change so that they can drop their commitments; A then runs a new auction to allocate new resources to the new path.

The commitment net protocol is summarized in Figure 2. Each lattice-like structure corresponds to a single announce-bid-award cycle. Time is represented on the vertical axis, while the horizontal axis lists the agents in the system, $\{N1, N2, \ldots, N10\}$. A set of new tasks is shown at the top of the figure, represented as task segment-location pairs, where a location is a reference to a point in space and time. A negotiation cycle consists of six steps: (1) computation of an appropriate list of bidders based on the projected target cone, (2) task announcement to the bidders, (3) bid computations by each of the bidders, (4) bidding, (5) winner determination, and (6) award. If the agents have additional time, each bidder can initiate subauctions, shown in the inset, to possibly improve the initial bids.

We are exploring several approaches for controlling

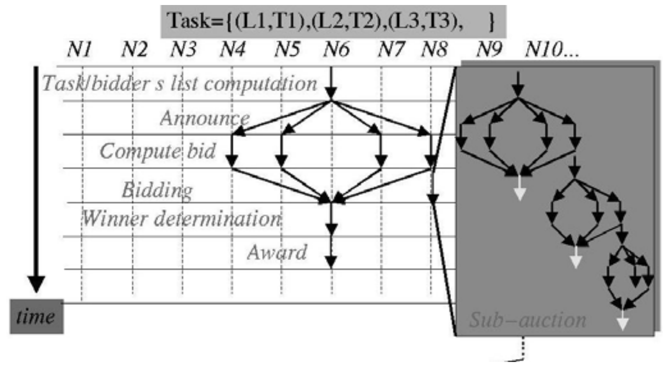

Figure 2: Illustration of the commitment net protocol.

complexity. The commitment net protocol without combinatorial bidding is an anytime algorithm. When combinatorial bidding is allowed, we make use of the iterative deepening algorithm proposed by (Sandholm & Lesser 1997), but with only single-task subauctions. With combinatorial bidding and multistage auctions, we are applying statistical and machine learning methods to analyze the relationships between cost of computation and task performance (Zilberstein & Russell 1995); we discuss these ideas further below.

Other methods for controlling complexity include the path projection step, which leads to better allocations, as discussed in the next section. One potential problem of system dynamics is the following. During computation of potential bidders, only a subset of the entire sensor web is considered; this means that care must be taken when initiating auctions to a potential coalition that might interact in a negative way with an overlapping coalition. To handle such problems we are informing winning bids of the other agents in the coalition, so that information is available to subsequent auctions.

Preliminary results

Figure 4 shows the interface to the multisensor tracking simulation that we have been using.[2] The experiments ran on a 16-node configuration with a single node performing track synthesis and reporting the results to the others. They lasted 90 simulated seconds, but because the processor load was so high, this required approximately 1 hour of real time, including initialization. Figure 3 reports on experiments regarding the benefits of path projection and Figure 5 reports on experiments involving multistage negotiation.

In order to isolate the difference between approaches, the listed message counts only cover messages related to the execution of a cycle. These include solicitations for participation and winner announcement. In addition to such messages, the system produces sector reports, target position reports, and TCP acknowledgments or

[2]The simulation is called Radsim and was developed by the Air Force Research Laboratory in Rome, New York, for the DARPA ANTS program.

	Avg Error(ft.)	Auction Msgs
Path-Projection	1.02	144
Non-Projecting	11.88	720

Figure 3: Evaluating path projection. Error is measured in feet (average distance between measured position and actual position).

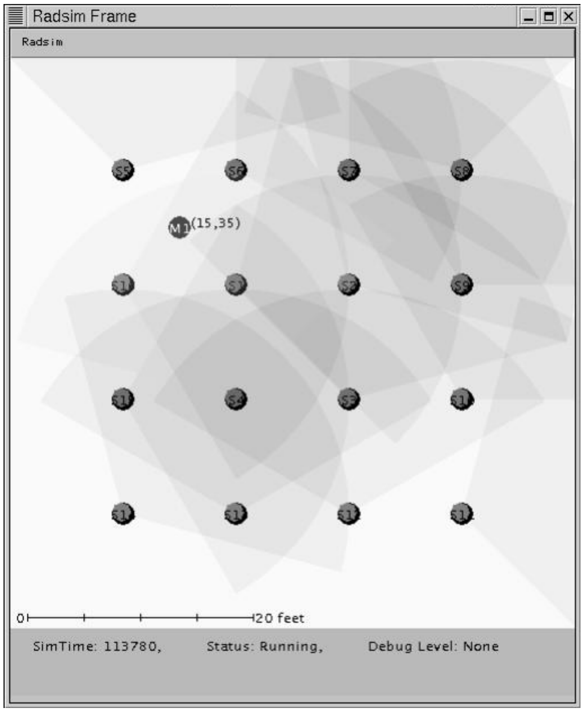

Figure 4: multisensor tracking simulation. Each small circle in the grid represents a sensor node; the active sectors of each node are highlighted. A target, labeled $M1$, is shown in the upper left corner.

re-sends. The number of such messages varies arbitrarily, and is comparable between approaches.

To satisfy an external constraint that power consumption remain less than 25%, we required that in all cases beam usage be limited to the four winners of each auction. Hence, at any point in time only four sectors were activated, resulting in a maximum expenditure that was within the power constraints.

Evaluating target path projection

In the algorithm allowing path projection, auctions governed 10-second blocks of future activity, with each block divided into 2-second increments during which agents were committed to a single action. Auctions were initiated as early as 8 seconds ahead of time, and winner determination occurred 2 seconds before the auction's results were to take effect.

The nonprojecting auction's high error rate reflects the fact that it lost the target for a large stretch of time. Because it was auctioning off only a single point in the future, it was subject to reactivity shortcomings where the target was already past the projected point by the time an auction had terminated.

On the other hand, the path projection algorithm achieved better accuracy by insulating itself from this problem. Even if a node did not receive its instruction before the target had reached the beginning of the auctioned segment, it was ready to act for the next 10 seconds, and could not fall behind permanently. That there was any error at all reflects upon shortcomings of the tracker module. In all cases the four closest nodes were selected to fire their correct sectors, and they always did so within 1 or 2 seconds of the target's arrival into the projected segment.

Not surprisingly, path projection was also able to save on message traffic since larger blocks of time were governed by each auction, and the number of auction messages is directly proportional to the number of auctions.

Evaluating subauctioning

Both algorithms ran over a system where four nodes were denoted to hold simulated commitments conflicting with any possible tracking assignment. That is, whenever they placed a bid reflecting their utility in participating in a track, they subtracted an additional sum. This would happen in cases where a node is committed to a conflicting task, and is adverse to dropping it. Our system recorded the number of times a node was forced to drop a commitment, resulting from the forced assignment of a tracking task.

When subauctioning was allowed, committed agents were able to pass on their tasks to neighboring nodes that were not reached by the initial announcement. Such neighbors would not be reached initially as a result of their greater distance from the projected segment, and their consequently lower utilities for participating. However, because of the commitment-dropping penalty, the neighbors do in fact have a higher utility for tracking, and wind up winning the subauction. As a result, few commitments were dropped.

When auctioning for a segment was allowed to take place only in one stage, the committed nodes were often forced to participate in auctions despite their penalties, as they were unable to contact neighboring replacements. As a result, many more commitments were dropped than with the subauctioning algorithm.

On the other hand, this naturally resulted in a smaller number of auctions, as reflected in the message count. However, the greater track accuracy achieved by the one-stage auction is not really an advantage, but actually a result of using committed nodes that were closer to the projected target than their otherwise optimal neighbors. That is, any gains in accuracy were at the expense of dropping commitments that, in the simulation, ought not to have been dropped.

In particular, the diminished accuracy did not result from poor reactivity or time delays associated with subauctioning, because subauctions were smaller than normal auctions, requiring announcement to fewer nodes.

	Lost Commits	Avg Error	Msgs
Subauction	2	2.41	198
One-Stage X	24	1.88	166

Figure 5: Evaluating multistage auctions (first line) versus single-stage auctions (second line).

Hence, they were able to execute in parallel with the main auction's most time-consuming activities, such as bid collection and winner determination. As a result, the subauctions often terminated before the main auctions would have finished anyway.

Task contention We have conducted some preliminary experiments involving two targets with task contention. In our preliminary experiments, we obtained an accuracy of 4.20 feet for one target and 4.18 for the other. Task contention occurs when some set of nodes is committed to an existing task, and a new task — occurring at more or less the same time — conflicts with that commitment, in the sense that the second task would require activation of an adjacent sector for tracking (recall, that at most one sector can be active at one time). One solution to this is for a node to alternate between sectors. However, a node has direct access only to local information; it must interact with its neighbors as otherwise it has no way of knowing that another node could not do a better job of tracking the second target.

Data acquisition for learning

The following describes our planned experiments to acquire performance profile information for associating problem types with computational requirements. That information will support the deadline-based commitment network algorithms.

Here is a formulation of the problem:

1. Agent A identifies a set of tasks (e.g., track segments for a projected target) and announces them to agents $B_1, \ldots B_n$, along with a deadline T time units into the future.
2. $B_1, \ldots B_n$ now have T time units to compute their own bids and/or perform a secondary auction.
3. Each B_i applies an Auction Time Allocation Method (ATAM) to decide how to allocate the T time units
4. The ATAM outputs a percentage P (greater than 0.0, less than or equal to 1.0). (1-P)*T units are allocated to a secondary auction (which is performed by recursing to step 1 with agent B_i in A's role), and then the remaining $P*T$ units are used to compute a final bid.

This scheme assumes no iteration (i.e., no re-bids). Also, note that the first $(1-P)*T$ units, while agent B_i is waiting for the secondary auction results to return, could be used to compute a local bid (i.e., assuming that the secondary auction fails).

We can treat the creation of the ATAM as the problem of learning a mapping *from* the problem characteristics of B_i's bidding situation *to* a bidding time percentage, P. Given the above scheme, the four major challenges are

1. Creating an anytime bidding algorithm that provides satisfactory results given a specified amount of time.
2. Identifying appropriate problem characteristics to characterize the time allocation problem.
3. Measuring the quality of an auction outcome (necessary for feedback to the learning algorithm, or to measuring the performance of a hand-designed algorithm).
4. Finding or designing an appropriate learning algorithm.

The following are potentially relevant problem characteristics. These are the aspects of the agent's situation and bidding context that affect the optimal allocation of time to the auction stages.

1. Total time available (i.e., T).
2. Number of agents bidding in the current stage.
3. Total number of agents in the "world."
4. Some measure of "bidding capacity" or "bidding likelihood" of the remaining agents (i.e., the agents who might be solicited for secondary-stage bids). One can envision many ways to estimate this. For example, some measure of the minimum or average "distance" of the secondary agents to the tasks to be bid on, or their "capability" to perform said tasks.
5. Cost and/or reliability of messages (impacts how valuable a secondary stage auction is).
6. Estimated time for current agent to compute an optimal bid (may want to focus on this rather than wasting time advertising a secondary auction).
7. Estimated capability of current agent to perform the task (if this is low, should focus on secondary auction).

We will gather data for (3) by setting up similar initial conditions, and then varying the problem characteristics and P (e.g., selecting randomly). We will measure the outcome (success or failure, or some quality measure of the result of the auction).

For (4), we will first use baseline hand-designed mappings. We then plan to consider regression algorithms. (This is a regression problem, rather than a classification problem, because the output is a continuous variable P, rather than, say, a binary class label. We will consider classification problem variations obtained by discretizing P – instead of predicting a continuous value for P, just break P into, say, three discrete values: 1 (no secondary auction), 0.2 (focus on secondary auction), 0.5 (evenly distribute time between primary and secondary auctions).

1. `Generate an initial allocation (e.g., by sequential auction)`
2. `Initialize CG, an unconnected graph with` m `vertices, each corresponding to a task`
3. `Iteratively improve the allocation as follows:`
   - `Add an edge that connects two unconnected subgraphs`
   - `Optimally allocate the tasks that correspond to the edges in the newly connected subgraph`

Figure 6: Anytime algorithm for task allocation. The algorithm can be interrupted at each iteration of Step 3.

Anytime combinatorial task allocation

We have been studying the effect of task interaction in resource allocation methods of the sort that we have described. Basic-level tasks in the multisensor tracking domain can have positive or negative interaction. For example, if two targets appear in succession at a particular sector of a node, then the cost of tracking each target in succession is lower than the cost of tracking each target independently: the reason is that there is a warm-up cost associated with the use of each sector. If targets appear one after another, then the warm-up cost is saved for each subsequent target. Conversely, it is impossible for an agent to simultaneously monitor two targets, each of which is in a different sector. The cost of performing both tasks is infinitely large, whereas the cost of performing either alone may be reasonable.

The possibility of task interaction in the sensor domain suggests that a combinatorial method for task assignment may be beneficial because sequential or parallel methods could result in inefficient or even impossible allocations. There are two main problems in applying a combinatorial auction mechanism. First, each agent is faced with a bid generation problem, in which it must compute all relevant bids for a set of tasks. However, in previous work we have shown that the number of relevant bids may be $O(2^m)$. Second, the agent acting as auctioneer must run a potentially costly winner determination algorithm. Because of these two problems, the granularity (i.e., frequency) of task allocations using the combinatorial mechanism can be low. In experiments on a 16-node simulation, tasks were assigned only every 10 seconds, in large part because of the communication and computational complexity of implementing the combinatorial auction.

Our anytime algorithm, called *Incremental Task Allocation Improvement* (ITAI), does not require a bid generation phase as input. Agents incrementally reveal their costs for bundles of tasks.

The algorithm is summarized in Figure 6. One way of performing the initial allocation in Step 1 quickly is by sequential auction. As discussed above, task interaction may lead this allocation to be suboptimal.

The task connection graph initialized in Step 2 directs the improvement phase of Step 3. At each iteration of the improvement phase, one edge is added to connect two unconnected subgraphs. For example, on the first iteration, an edge is added between any two of the vertices. On the second iteration, an edge may be added between two other vertices, or between one other vertex and one of the two previously connected vertices (thus creating a connected 3-vertex subgraph).

On each iteration, an optimal allocation (e.g., by a combinatorial auction with optimal winner determination) is made for the tasks corresponding to the newly connected subgraph. The procedure terminates when CG is connected (i.e., adding an edge cannot connect two unconnected subgraphs). The algorithm is anytime because it can be stopped at any point during the improvement phase and can return the lowest cost allocation attained so far.

To generate the initial allocation, an agent need reveal only m costs, one for each initial task allocation. In the improvement phase, even if a combinatorial auction algorithm is used, an agent is initially faced with a much simpler bid generation problem, because the algorithm is run over only a few tasks. If an exhaustive enumeration of task allocations is used instead of a combinatorial auction in that phase, the bid generation problem is replaced by incremental revelation of costs for sets of tasks.

Theorem 0.1 *The algorithm is guaranteed to find the optimal task allocation in the final iteration.*

Proof: In the final iteration, CG is connected. If the algorithm then optimally allocates all m tasks corresponding to edges in CG, the allocation will be optimal. ∎

Time complexity Similar to the general iterative deepening search algorithm, ITAI incrementally expands the scope of its search for the optimal task allocation. The time spent on task allocation is the sum of the time spent generating the initial allocation, plus the time spent improving the allocation. As Theorem 0.2 illustrates, the complexity of the algorithm is the same as an algorithm that performed optimal allocation of all tasks in a single step.

Theorem 0.2 *Assuming an iteration of the improvement phase that allocates i tasks takes $O(n \cdot 2^i)$ time, the running time of the improvement phase is $O(n \cdot 2^m)$.*

Proof: The maximum number of improvement steps results if a single vertex is connected to the subgraph at each iteration of Step 3. In this case, there are $i = 3Dm - 1$ steps, with the numbers of connected vertices running from 2 to m. The total running time of the improvement phase is

$$O(\sum_{i=3D2}^{m} n \cdot 2^i) = 3DO(2n(2^m - 2))$$

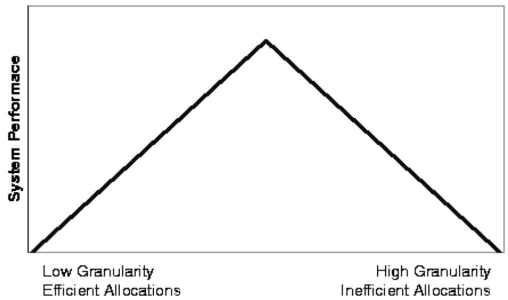

Figure 7: Expected ANTS system performance using ITAI.

which is $O(n \cdot 2^m)$. ■

We have been experimenting with the ITAI algorithm to determine the tradeoff between finding an efficient task allocation, and finding a good (but possibly inefficient) allocation quickly. Our current experiments are targeted at uncovering the tradeoff between achieving higher granularity and efficiency of task allocation. Figure 7 illustrates our hypothesis that with a very high granularity (i.e., frequent task allocations), the allocations will often be inefficient (i.e., tasks will frequently be assigned to agents that are not well suited to performing them), and tracking performance will be low. Similarly, with very low granularity (e.g., 10 second intervals in the current system), even though efficient assignments are made, tracking may be poor because the frequency of data collection is low. ITAI allows us to investigate the middle ground, where granularity is between the two extremes, and allocation efficiency is suboptimal but good. We believe that optimal tracking performance will be attained with this intermediate level of granularity.

Large-scale resource allocation

To address scalability — on the order of hundreds of sensors and targets — we have developed an agent-based computational model of semicentralized task allocation. This is called the Distributed Dispatcher Model (DDM). In this model, centers are associated with geographical areas, called districts. Centers are organized hierarchically and each center is responsible for a particular district. Each level of the hierarchy controls the level below it. We refer to each center as a coalition leader. In the design of the DDM we had two goals. The first goal was to generate a map of targets as a function of time. The second goal was to achieve good coverage of sensors over the area of interest. Testing of the model was done using a variation of the multisensor tracking simulation described earlier in which sensors are mobile and there is no need for clock synchronization or very close cooperation between the sensors.

Several entities appear in the simulation used by the the DDM system. The first is a target. A target is created outside of the controlled area. A target has the properties of location and velocity. Each target moves in a steady speed; however, it may change its velocity from time to time.

The second entity is a mobile Doppler sensor. We make use of a constant number of Doppler sensors to map the targets in the controlled field. Each sensor has the capability to activate three beams, one at a time, to track targets in its detection range. A sensor acquires the amplitude and the radial velocity of each target located in its detection range. Unlike the simulation described earlier, a sensor takes four consecutive measurements over a short interval of time. The sensor uses an internal clock to tag each measurement with a time. Although all the clocks in the system are periodically synchronized; time differences might arise between synchronization points.

The third is the sampler. To every sensor is attached a sampler. The sampler supplies sets of target states $\{t, x, y, vx, vy\}$ according to its assigned sensor and commands the sensor in a particular direction and with a particular speed. We have successfully used one sampler to determine a set of locations of a sensed target, using four consecutive samples of the sensor and an assumption that the target does not change its velocity during this time.

The fourth is the coalition leader. The coalition leader controls a particular area. Each higher level will control a larger area. We distinguish between two types of coalition leaders: a zone coalition leader and a sampler coalition leader. Whereas the zone coalition leader controls other coalition leaders, the sampler coalition leader controls the behavior of a set of samplers in a given area and is responsible for directing sensors. The main task of both is to form a map of their areas. Another important task of a zone coalition leader is to balance the number of sensors in its area.

The main difference between a zone coalition leader and a sampler coalition leader is that the first is responsible for an area made up of different zones while the latter is directly responsible for the behavior of the sensors in a specific zone. Therefore, most of the zone coalition leaders may be in charge of other zone coalition leaders at a lower level. The lower zone coalition leaders are in charge of sampler coalition leaders. As mentioned above, the algorithms for zone coalition leaders and sampler coalition leaders are different. Whereas the zone coalition leader should balance the number of sensors between zones and should decide how many will pass from one zone to another, the sampler coalition leader should follow orders from its superior zone coalition leader and decide to which sensor to pass and how to accomplish that. A sampler coalition leader should also make sure to direct the sensors in the zone of its responsibility.

Figure 8 illustrates a hierarchical representation of the distributed solution.

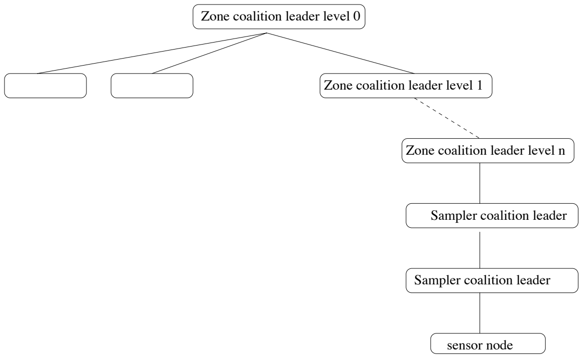

Figure 8: Distributed solution hierarchy.

Forming the state map algorithm

The first algorithm uses a hierarchical structure to achieve global information over the controlled area. Each coalition leader has global information of its controlled area, whereas the top-level coalition leader (level 0) has global information over the entire area.

Each coalition leader implements the following algorithm:

1. Update the current knowledge base every second using known information about all targets in the controlled area. The format of the items of information is $\{t, x, y, vx, vy\}$ for each target. This information may result in a new time-dependent state map.
2. Every δ seconds ask each subcoalition leader or sampler for new information.
3. Filter redundant information about the same target and begin again from (1).

Controlling the sensors

The second algorithm uses the knowledge base stored in each coalition leader to control the sensor's behavior. The motivation is to balance the ratio of the number of resources (sensors) over all zones with the ratio of the targets.

Each coalition leader implements the following algorithm:

1. Every δ seconds ask the superior coalition leader (if one exists) for a prediction about incoming targets to the controlled area in $t + \delta$.
2. Ask the superior coalition leader (if it exists) for instructions to send sensors involving movement to a neighboring zone. The neighboring zone will be at the same level.
3. Form a predicted knowledge base for $t+\delta$ according to the known data (first algorithm) and the prediction from the superior coalition leader.
4. Based on its knowledge base and the prediction obtained from the superior coalition leader, form a set of orders to be sent to its subordinate leaders. The coalition leader will build this set to balance the ratio of the number of sensors over all controlled zones with the ratio of the targets. The coalition leader will calculate how many sensors should be in each zone. This will be determined by summing all the targets, T, and sensors, S, in the area. The number of sensors in each zone will be the number of targets in the zone multiplied by the ratio S/T. The coalition leader will use the number of sensors that should be in each zone and the number that is actually in each zone to find the difference. The difference represents the number of sensors that should be passed to the zone. The coalition leader will generate an order to move sensors from a negative difference to a positive one. If the coalition leader is a sampler coalition leader, it will follow any orders to direct sensors to an indicated zone.

Sampler coalition leader behavior

A sampler coalition leader receives instructions from its superior coalition leader regarding how many sensors to shift and to which neighboring zone to shift them. The sampler coalition leader determines which sensors to pass to the needed zone and how to accomplish that. The sampler coalition leader chooses the closest sensors to the needed zone and orders them to move at a speed that will pass them to the next zone in time $t+\delta$ where $t+\delta$ is the time that its superior zone coalition leader wanted those sensors to be at the next zone. The direction is calculated so the sensor that will pass into the same zone will pass at an equal distance between them. For instance, one sensor will pass at the middle of the border, and two sensors will pass the points at one third and two thirds of the border. Calculating the intersection points, the velocity and the direction is very simple. By knowing how many sensors should move to the next zone, S, and the length of the borders between the zones, L, the coalition leader may determine that every L/S meters a sensor should pass to the next zone. The leader will then allocate a sensor to each intersection point to the closest sensor. By having the current location and destination of the sensors, the velocity and the direction can be derived.

A sampler coalition leader should also guide the movements of the sensors that stay in its zone. That can be accomplished by forming milestone points over $t + \delta$ time units for each sensor. These points will represent a route and a speed for the sensor. The sampler coalition leader will generate the points by forming a quick prediction of the movement in the interval t to $t + \delta$. The purpose of the movements will be to keep the sensors moving in straight lines and at the same speed until they cross the detection zone of other sensors. In that case, the sensors act as if heading toward a collision and change direction. The coalition leader can determine appropriate milestones by activating a quick simulation from t to $t + \delta$ of the sensor movements; the coalition leader has all the necessary information about the sensors and targets in its area.

Fault tolerance

The DDM is fault tolerant in the following way. If a coalition leader does not receive a response from one of its subordinates, it will record the fact that the subordinate is not available (dead) and it will then divide that subordinate's area between neighboring subordinates. If the dead subordinate comes back to life, its original sector will be returned to it.

Since the coalition leaders in the zone that was divided should know who their new leader is, the coalition leader (that divided the zone) should know who is in that zone. To handle this complication we are experimenting with another approach for choosing one of the members in the zone to be the new leader, and informing the others rather than changing the size of the zone.

Verifying system dynamics

We briefly describe a logic — to be reported in detail elsewhere (Ortiz 2001) — that we have developed for characterizing various useful domain-independent societal behaviors. The logic makes use of the notion of counterfactual dependency — roughly, that some system property such as stability holds because if some event (such as a perturbation) were, counterfactually, to occur, then the system would eventually move back to its original state. In many cases, it is useful to carry through such an analysis within spatially and temporally bounded areas of subsystem behavior. This requires that one have some way of analyzing a subsystem while assuming that the rest of the system deviates from actual behavior as little as possible; the notion of most similar world is one that plays a central role in counterfactual logics. The full paper also presents a formal definition of what it means for a behavior to be *emergent* and demonstrates how agent designs can be made to computationally exploit beneficial emergent behavior.

Some of the sorts of group behaviors that we have explored within the logic described have their origins in control theory and include notions of system equilibrium, system stability, and system trajectory. Unfortunately, control theory is intended for systems that can be described in terms of sets of differential equations; this is not always possible. The logic we describe is more appropriate for systems that are usually described in the distributed systems literature as discrete event systems.

Suppose we have a group of agents, G, embedded in some environment, Σ. For the multisensor tracking application, G will consist of a set of sensor agents and Σ will describe a communication system linking the sensors together with descriptions of moving objects that enter G's field of view. The goal of G is to track the moving objects by directing sensors toward the moving objects. We are interested in various dynamic, global properties of G as described below. Some of these might be desirable while others are to be avoided.

System trajectory It is useful to be able to capture the intuition that a system is "moving" in some particular direction along some trajectory. This can be done by identifying some measure of progress toward some goal state, ϕ. To say that G is moving in the direction of ϕ is not just to say that G will reach ϕ but that it is also making some progress toward ϕ, that is, that G is changing state along a trajectory in which it will become closer to ϕ, according to the chosen measure. Defining a trajectory away from some state can be done in an analogous way.

Equilibrium region System G will be said to be in *equilibrium* over the interval $[t_1.t_2]$, with respect to condition ϕ and some set of potential environmental events, E, just in case G is executing some strategy (typically, a set of conditional actions) over $[t_1, t_2]$, which will maintain the truth of ϕ, no matter what hypothetical event, $e \in E$, might occur (Figure 1 illustrated such an e). The condition ϕ might be a complex formula that also places some bounds on the condition; for example, ϕ could stand for "throughput of the system is greater than b."

Stable equilibrium System G is a *stable equilibrium* over $[t_1, t_2]$ and with respect to some set of potential environmental events from some set, E, some condition ϕ, and some associated maximum distance from ϕ, just in case G is in equilibrium and there is some event $e \in E$, such that if e were to occur: (a) G would move away from ϕ, but (b) after some interval of time or distance less than b, G would move on a path that would eventually take it to ϕ.

Unstable equilibrium Subsystem G is an *unstable equilibrium* over $[t_1, t_2]$ and with respect to condition ϕ, potential environmental events E, and distance d, just in case G is in equilibrium and there is some event, $e \in E$, such that if e were to occur G would move away from ϕ past b and never return to ϕ.

Spatio-temporal dependency Often, it is useful to identify dependencies between spatio-temporal regions of activity. For example, one might say that $G' \subset G$ is supporting some $G'' \subset G - G'$ just in case there is some activity, α, that represents the behavior of G' and some later β that represents the behavior of G'', and if α had not occurred then β would also have not occurred. In certain instances it can be useful to recognize that some region of activity depends on another in some positive sense (that is, it is being "helped" by) or in some negative sense (it is being "hindered" by).

Emergent behavior Roughly speaking, an activity α of group G is said to emerge from the activity β of G, both occurring over the same interval $[t_1, t_2]$, just in case if β had not occurred then α would not have occurred either. Typically, β requires less information or knowledge about the environment than α. In addition, one is often interested in uncovering a particular β that would computationally exploit some feature of the environment: that is, require less computation than α.

Our examples motivate the need for a counterfactual analysis. For instance, the definition requires that one have some way to take a theory describing the execution of an actual system and then consider the consequence of introducing a perturbation. Since the perturbation conflicts with what actually happened, one needs to modify the original theory in some minimal way to maintain consistency. A second difficulty involves the following. Suppose an activity is described in terms of the temporal sequence of events e_1, e_2, e_3, and e_4, each of which can be thought of as having caused the next event in the sequence. Suppose also that e_1 caused another sequence beginning with e_5. If we then consider the counterfactual consequences of, say, e_2 not occurring, then we need to make certain that e_1 nevertheless occurs. Otherwise, e_1's not occurring might affect the secondary sequence that begins with e_5, resulting in a spurious connection between the nonoccurrence of e_2 and the secondary branch. This inherent *temporal assymetry* of counterfactuals is discussed in greater detail in (Ortiz 1999) and in the full paper.

Summary

The class of solutions that we have presented and that are the subject of ongoing development were designed with several requirements in mind. In the first place, the solutions were required to be distributed: they could not rely on a centralized coordinator for assigning resources to tasks; such solutions would lack any measure of fault tolerance. The sequential, combinatorial, and multistage auction derivatives that we discussed satisfy that requirement.

The sorts of problem that have been the focus of our work further introduced the requirement for a real-time solution; we have approached that constraint from two directions. We are first developing anytime algorithms for both sequential and combinatorial allocation. The case of multistage combinatorial auctions poses a special problem as intra- and inter-agent interaction costs must be balanced; to address that issue we are drawing on machine learning techniques to associate problem types with computation profiles. From another perspective, we are exploring various organizational structures that can balance processing loads in large-scale implementations.

Communication is a resource that must also be managed; in preliminary experiments we demonstrated the advantage of persistent bids in reducing message traffic. We believe further experiments will support that claim as well as demonstrate fault tolerant behavior through the reciprocal background commitments we described. All these are issues important to system dynamics; we hope that the logic that we have briefly described can serve as a tool with which system designers can verify that an agent society performs in the desired manner.

Acknowledgments

This research was funded by DARPA Contract F30602-99-C-0169 under the Autonomous Negotiating Teams (ANTS) Program.

References

Grosz, B. J., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(1):269–357.

Hunsberger, L., and Grosz, B. J. 2000. A combinatorial auction for collaborative planning. In *ICMAS*.

Kraus, S. 1996. An overview of incentive contracting. *Artificial Intelligence Journal* 83(2):297–346.

Ortiz, C. L. 1999. Explanatory update theory: Applications of counterfactual reasoning to causation. *Artificial Intelligence* 108:125–178.

Ortiz, C. L. 2001. Provable emergent properties of agent societies. in preparation.

Sandholm, T. W., and Lesser, V. R. 1997. Coalitions among computationally bounded agents. *Artificial Intelligence* 94((1-2)):79–98.

Sandholm, T. W. 1999. Distributed rational decision making. In *Multiagent Systems*, 201–258. MIT Press.

Sen, S., and Durfee, E. H. 1996. A contracting model for flexible distributed scheduling. *Annals of Operations Research* 65:195–222.

Smith, R., and Davis, R. 1983. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence* 63–109.

Zilberstein, S., and Russell, S. J. 1995. *Approximate Reasoning using anytime algorithms.* Kluwer Academic Publishers.