Combinatorial Auctions for Resource Allocation in a Distributed Sensor Network [*]

John Ostwald and Victor Lesser
UMass Computer Science Technical Report 04-72

August 31, 2004

Abstract

This paper discusses a solution to the problems posed by sensor resource allocation in an adaptive, distributed radar array. We have formulated a variant of the classic resource allcation problem, called the *setting-based resource allocation problem*, which reflects the challenges posed in domains in which sensors have multiple settings, each of which could be useful to multiple tasks. Further, we have implemented a solution to this problem that takes advantage of the locality of resources and tasks that is common to such domains. This solution involves translating tasks and possible resource configurations into bids that can be solved by a modified combinatorial auction, thus allowing us to make use of recent developments in the solution of such auctions. We have also developed an information-theoretic procedure for accomplishing this translation which models the effect various sensor settings would have on the network's output.

1 Introduction

The CASA project, funded by NSF, is constructing an adaptive, distributed array of radar dishes to monitor tornados and other meteorological phenomena. The radar dishes have a number of parameters that can be adjusted, such as what area they are sweeping across, pulse rate, etc. The resource allocation problem in this domain is, loosely stated, to decide at each timestep

[*]Work supported primarily by the NSF Engineering Research Centers Program under NSF Award Number EEC-0313747001. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

(allocation cycle) what setting to have each sensor in, taking into account the needs of various monitoring tasks. (See Section 2 for a more precise statement of the problem.)

For example, consider the sensor layout shown in Figure 1. In this diagram S1 and S2 denote radar dishes, and H1 and R1 are areas of interest for a hail monitoring and a rainfall monitoring task respectively. In choosing an allocation for this timestep, a resource allocation process would have to make a number of decisions. For instance, it would have to decide whether each sensor should sweep over one task's area, or both should sweep the same task if one is more important, or whether one or both of the sensors should sweep over an area large enough to encompass both tasks. It would also have to decide whether to optimize the other parameters for each radar in a manner that is useful to the rain task, the hail task, or both. In short it would have to look at all the options and consider all the preferences provided by the tasks, and choose the globally best settings.

The rest of this paper is structured as follows: In Section 2 we present our resource allocation problem formally and compare it to the classic resource allocation problem. In Section 3 we compare our problem and approach to those presented in various well-known papers on resource allocation. In Section 4 we present the details of the system we built to solve the problem in the CASA domain. In Section 5 we discuss the performance of our system, and finally, in Section 6, we will discuss the significance of our work and what we plan to do next.

2 Problem Overview

The problem we are addressing, which we will call the setting-based resource allocation problem, is closely related to the classic resource allocation problem. We will present the classic problem first, for comparison, then introduce the setting-based variant. In a classic resource allocation problem, we have a set $\mathbf{R} = \{r_1, r_2, ...r_n\}$ of resources and a set $\{t_1, t_2, ...t_m\}$ of tasks. Each task has a *task utility function* $T_i : Powerset(\mathbf{R}) \to \mathbb{R}$ representing the utility that the task would yield for each set of resources that could be allocated to it. The resource allocation problem is then to choose, for each task t_i, a set S_i of resources to allocate such that the S_i's are disjoint and $\sum_{i \leq m} T_i(S_i)$ is maximized.

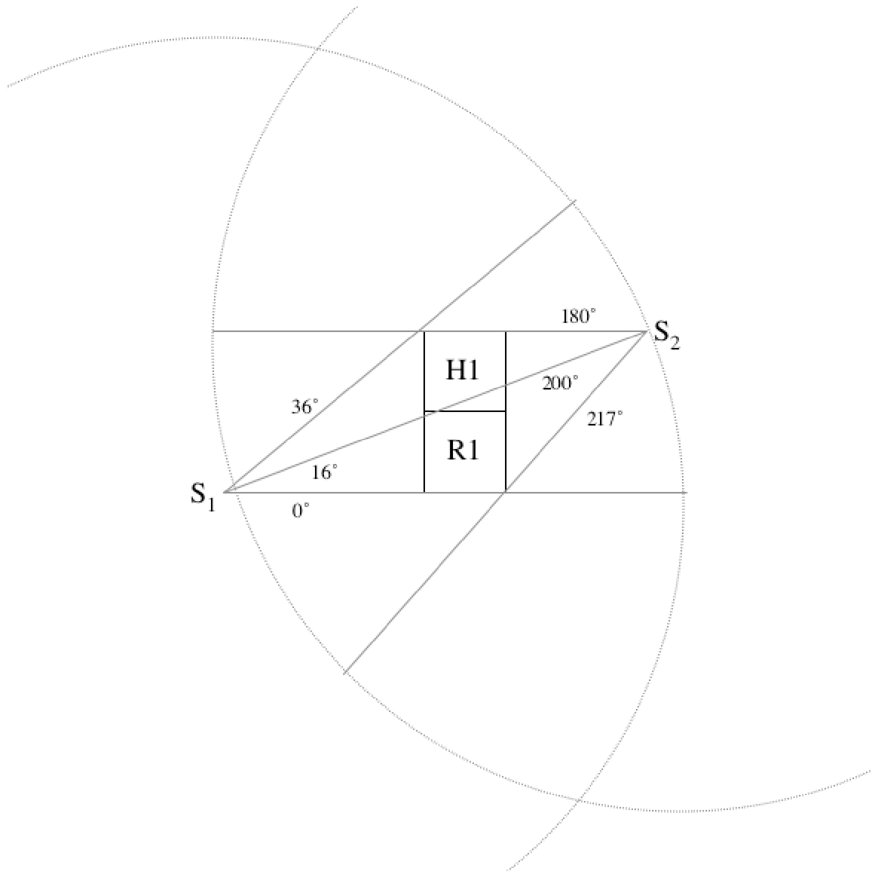


Figure 1: Example Arrangement of Sensors and Tasks

In the setting-based resource allocation problem, we have a set of resources $\mathbf{R} = \{r_1, r_2, ...r_n\}$ each of which has a set $\mathbf{V_i}$ of settings it can be in. We will define a *global configuration* to be a mapping from the resources to the elements of their respective $\mathbf{V_i}$'s, specifying what setting each resource is in. We will let $\mathbf{C}$ denote the set of possible *global configurations.* There is also a set $\{t_1, t_2, ...t_m\}$ of tasks. As before, each task has a *task utility function* $T_i : \mathbf{C} \rightarrow \mathbb{R}$, but in this domain the function evaluates *global configurations* instead of sets of resources. The problem is to choose a *global configuration* c such that $\sum_{i \leq m} T_i(c)$ is maximized.

The intuitive approach to solving this problem is simply to consider every possible *global configuration* and to evaluate every task for each. If we address the problem in this manner, the number of *configurations* that will have to be searched through is $\prod_{i \leq m} |\mathbf{V_i}|$ and the number of *task utility functions* that will have to be evaluated is $m \prod_{i \leq m} |\mathbf{V_i}|$.

An alternate approach, and the one we will be focusing on in this paper, requires us to augment the language in which the *task utility functions* report their values. First, they need to be allowed to specify minimum values for *partial configurations*, that is specifications of settings for only some of the resources. The value for a *global configuration* is then the highest value specified for either it or any partial configuration whose settings agree with it's own. We will henceforth use the term *configuration* to refer to both partial and global configurations. Second, the *task utility functions* must be allowed to give a *configuration* a value of 0 by omission: each *task utility function* provides a list of *configurations* and values for those *configurations*, and any *configuration* not enumerated in the list is assumed to have value 0. We will refer to this list as a bid and the set of *configurations* evaluated in the bid as the *explicit domain* of the *task utility function.* It is worth noting that both of these augmentations make sense in the domain of a sensor network, where it is often the case that the value of a *global configuration* to a task depends only on the settings of a few resources (in a sensor net, these would be the sensors near whatever is being monitored by the task). Also note that we are not restricting the configurations that the tasks can evaluate or the values they can give them, but simply allowing tasks to evaluate classes of configurations if they so desire. A task could still evaluate every *global configuration* individually if the evaluations of those *configurations* required it.

Our proposed approach, then, is as follows: For every task t_i, we evaluate T_i for every *configuration* in its *explicit domain.* We then find the *global configuration* that has the highest utility yielded the *task utility functions* of every task. The number of T_i's that will have to be evaluated in this approach is $\sum_{i \leq m} |explicit\_domain(T_i)|$. We then have the added challenge of finding the best *global configuration* considering all the bids, which amounts to a search through a space of size $\prod_{i \leq m} |explicit\_domain(T_i)|$. We call this approach the task-based approach.

```
$Bidset \leftarrow \emptyset$
for each task $t$ do
    let $T$ denote the task utility function of $T$
    for each set $S$ of resources do
        for each configuration $c$ of $S$ do
            if $c$ is in the domain of $T$ then
                $Bidset \leftarrow Bidset \cup \{c, T(c)\}$
            end
        end
    end
end
return bid-combiner(Bidset)
```

Algorithm 1: Pseudocode for Task-Based Approach

Clearly, the applicability of this approach hinges on two things: the size of the *explicit domains* of the *task utility functions* and the difficulty of combining the bids. We will first examine the size of the *explicit domains* and then discuss how we will combine bids. The size of the *explicit domains* can vary greatly depending on the specifics of the application. A task could be concerned with every possible combination of the settings of every resource (in which case the size of that task's *task utility function*'s *explicit domain* would be $\prod_{i \leq n} |\mathbf{V_i} + 1|$), or at the other extreme, it could be interested only in one setting of one resource. In the worst case, if every *task utility function* referred to every *configuration*, the number of evaluations of *task utility functions* would be $m \prod_{i \leq n} |\mathbf{V_i} + 1|$, which is asymptotically the same as in the intuitive approach as the V_i's increase. This means that, if we can address bid combination, this approach is as reasonable as the intuitive if the resources have a large number of possible *configurations*.

The task-based approach becomes even more reasonable for applications where tasks and resources are linked, that is for any given task, there will be relatively few resources that it will be interested in. This is the case when the tasks are highly specialized and only a certain class of resources is useful for each task or when the tasks and resources have a physical location and only nearby resources can be used for a task. Many distributed sensor networks, including the CASA array, have this last characteristic.

So, the main problem of this approach is clearly the bid-combination problem. We address this problem using recent innovations in combinatorial auction technology. A combinatorial auction is a silent auction in which bidders can bid on sets of items instead of single items. Each bidder provides sets of items and corresponding prices for each set, and the auctioneer chooses the set of bids that maximizes the payment. In the last few years great deal of progress has been made at efficiently solving these auctions (i.e. finding the optimal allocation). These algorithms can be modified to handle the bid combination aspect of our setting-based resource allocation problem, by considering the resources items and the tasks bidders (the role of the settings will be discussed in Section 4.4). The particular solution we are using is Tuomas Sandholm's BOB algorithm, which can solve auctions involving hundreds of items and thousands of bids in under ten seconds [5, 6]. The modifications we made to the BOB algorithm are discussed in Section 4.4.

3 Related Work

A great deal of research has been devoted to the classic resource allocation problem and variants thereof. However, to our knowledge, no work has been done on the setting-based resource allocation problem. The classic problem can be viewed as a simplified setting-based problem in which each resource setting represents allocation to a single task. More precisely, the simplifications of the classic problem are 1) that every resource has the same set of possible settings, 2) that the number of settings in this set is the same as the number of tasks, 3) that the value offered by each task's *task utility function* depends only upon which resources are in one specific setting (the setting corresponding to being allocated to that task), and 4) that the setting that is relevant to each task is relevant to no other task. This section will examine several well-known papers that deal with the classic resource allocation problem and examine how they differ from our work beyond their addressing of the classic problem.

In [7], the resource allocation aspect of the problem they are addressing is simplified by the fact that their *task utility functions* fall into two types. Tasks of the first type are interested only in sets of size one, which means that if there are n resources, the domain size of any *task utility function* is n, whereas if they were allowed to refer to any set of resources it would be 2^n. The other type of *task utility function* they have has only two values in it's range: zero and another (implicit) number. This means that they don't need to take into consideration a task's preferences among sets of resources that will satisfy it.

In [2] the resource allocation problem is phrased as a *list coloring problem*, which is like the *map coloring problem* except that each node can take on only certain colors. This means that the *task utility functions* all refer to sets of size one, which as before decreases the size of the *task utility function* domains to the number of resources. Further, since a solution must satisfy every task, there is no concept of different sets having different values to a task, or of different tasks being worth more than others, which means that there are only two values that any *task utility function* can take on. Note that this is an even stronger restriction than the second in the last paragraph, since in that paper it could be worth more to satisfy one task than another.

[4], is addressing the problem of network congestion, and as such there is only one resource (the network) and the question is simply how many units of it each task (user) gets. This means that the *task utility functions* must have the same value for any set of resources of a given size.

[8] presents the results of a competition in which agents compete to purchase commodities. This means that their problem differs from our version of the resource allocation problem in that they are not trying to find a globally optimal solution. While this adds various strategic elements to the problem, it eliminates the need to find an optimal solution. They also limit their task evaluation functions in various ways in different portions of the competition. In the hotel booking portion of the competition, resources were allocated in a single-unit auction, which means that as before, the size of the *task utility function* domains is the number of items. In the entertainment booking portion of the competition, the *task utility functions* were either directly proportional in value to the number of resources (imposing the same restrictions to the *task utility functions* as in [4]), or proportional to the number of nights that the entertainment tickets covered (restricting the ranges of the task utility functions to five values).

4 Implementation

We have implemented the task-based approach for the domain of sensor scheduling in the CASA radar array. Here, the resource allocation problem is deciding what setting each sensor should be in, taking into account all the various tasks that are present at the timestep. Tasks, in this domain, consist of a request for a scanning of a certain type to occur in a particular area, along with data about the utility of the task. We estimate the value of various *configurations* of sensors to tasks and send these evaluations as bids to the combinatorial auction solver and get our final answer.

We will now address details of the implementation. In section 4.1, we will examine the exact format of the bids which are sent to the auction program; in section 4.2 we will examine how we evaluate *configurations* to form these bids, i.e. the exact nature of the *task utility functions* for our domain; in section 4.3 we will examine how we decide what settings to consider for each task, i.e. how we choose the *explicit domains* of our *task utility functions*; and in section 4.4 we briefly introduce the BOB algorithm and discuss modifications to the auction solver.

4.1 Bid Format

A bid for a specific task consists of some number of clauses, each clause representing the settings of some number of sensors. The settings consist of a start angle, an end angle, and a set of values for the rest of the parameters, corresponding to the kind of task they are optimized for (so the setting {S1: 0-30, R} would mean sensor 1 sweeping from 0 to 30 degrees, with the rest of the parameters optimized in the manner that is best for measuring rainfall).

For example, we could have a rainfall task whose bid was the following

```
R1:
{S1: 0-18, R; S2 200-217, R} -> 15 utils
XOR
{S1: 0-18, R; S2 180-217, RH} -> 6 utils
XOR
.
.
.
```

where RH is a set of parameters which does a decent job for both rainfall and hail. The meaning of this is that if, for example, the *global configuration* includes {S1: 0-18, R; S2 200-217, R}, then task R1 will yield 15 utils.

To present an example: recall the sensor layout shown in Figure 1 where S1 and S2 are sensors, and H1 and R1 are the areas of interest of a hail and a rainfall task respectively. The bids that the system generates for this *configuration* are listed in appendix A. The first three clauses of R1's bid are as follows:

```
R1:
{1: 0-36, R }:  11.46
XOR
{2: 180-200, H 1: 0-36, R }:  11.46
XOR
{2: 180-217, RH 1: 0-36, R }:  11.46
XOR
.
.
.
```

4.2 Bid Clause Evaluation

In this section we will examine how we evaluate a *configuration*'s value to a task, that is how we calculate *task utility functions*. We use an information-theoretic approach similar to [1]. The key idea is that the entire sensor-net system, of which the resource allocator we have built is one small component, is being asked to make a decision about some aspect of the state of the world, such as what the rainfall is in a particular region. Our *task utility functions* attempt to model the quality of the decision that the larger system is likely to make given a particular *configuration*. We make the assumption that we have a distribution over the state the environment could be in with respect to the decision to which the task relates. (e.g., this distribution would be over the possible amounts of rainfall in the area if the task were a rainfall task.) This distribution used for the examples in this paper is uniform, but could later take into account the system's knowledge. For each decision the system might make in each possible state of the environment, there is an associated utility. See figure 2 for a table of these values for rainfall. Our system estimates what decision would be made given each sensor *configuration* in each state of the environment and then decides the bid clause value based on the improvement in decision that *configuration* would yield. In other words, to decide how much a *configuration* is worth to a task, we consider the states the environment could be in and look at how good a decision the meteorological algorithms would make in each state.

To make things more concrete: the value of a specific *configuration* to a task is:

$$\sum_{E}\left[\sum_{M} P(E)P(M|E)Val(E,D(M)) - Val(E, defaultdecision)\right] \tag{1}$$

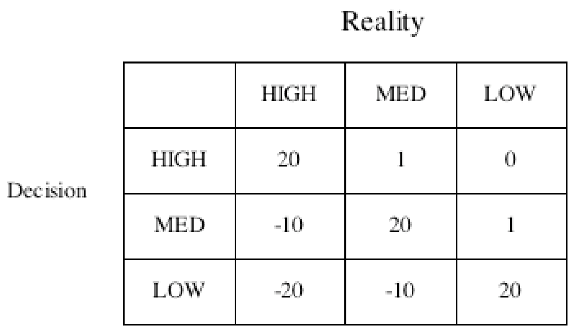
Where E is the current state of the environment, M is the set of measurements the sensors could make, $P(E)$ is the probability of the environment being in a state, $P(M|E)$ is the probability of taking measurements M in state E, $Val(E,d)$ is the utility of making decision d in state E, $D(M)$ is the decision resulting from measurements M and $defaultdecision$ is the systems current best guess at the state of the environment.

$P(E)$ is currently assumed to be 1 divided by the number of possible states but it could be based on knowledge from previous timesteps. $defaultdecision$ is currently hardcoded to be the middle value, but could also be based on information from previous timesteps, and $Val(E,d)$ is looked up in utility tables like the one shown in Figure 2. $P(M|E)$ and $D(M)$ are estimated using Bayesian networks, as is discussed in the next paragraph.

To calculate $P(M|E)$, we need to model the relationship between the state of the environment and the sensor readings. To calculate D(M), we need to model the relationship between the sensor measurements and the decisions that the meteorological algorithms will make. We use Bayesian networks to model both of these (see Figure 3). In fact, since the meteorological algorithms are themselves attempting to model the relationship between the sensors and the environment, we use the same Bayes net to model both the sensor readings and the decision making, inferring first down the tree, then up. Thus, for each state of the environment, we infer a distribution over possible sensor readings, and then for each set of readings, we run the inference back up the tree to find our decision.

```
$v \leftarrow 0$
$template \leftarrow retrieve\_bayes\_template(task, configuration)$
$update(template)$
**for** $e \in E$ **do**
    **for** $m \in M$ **do**
        $v \leftarrow v + P(e) \times template.getP(m) \times Val(e, template.decide(m))$
    **end**
    $v \leftarrow v - Val(e, d)$
**end**
**return** $v$
```

Algorithm 2: Value of a Specific Configuration to a Task



		Reality		
		HIGH	MED	LOW
Decision	HIGH	20	1	0
	MED	-10	20	1
	LOW	-20	-10	20

Figure 2: Utility Table for Rainfall

For an example, let us look at how the clauses of R1 in the example from Figure 1 were constructed. When the task arrives, the system retrieves a template for each number of sensors that could be used to satisfy the task (in this case 1, 2 or 3). Figure 3 is the template for rainfall using two sensors. The bottom level of the net refers to sensor measurements, and the top is the probability of the aspect of the environment we are supposed to be deciding about, in this case the amount of rainfall in the area of interest. It is important to note that these Bayes nets will never be used to actually draw inferences from the sensor readings—a sophisticated meteorological system will do the actual data analysis. Once the template has been retrieved, we update its conditional probability tables to reflect the current local signal to noise ratio. Then we do the calculation described in Expression 1 for each setting of sensors.

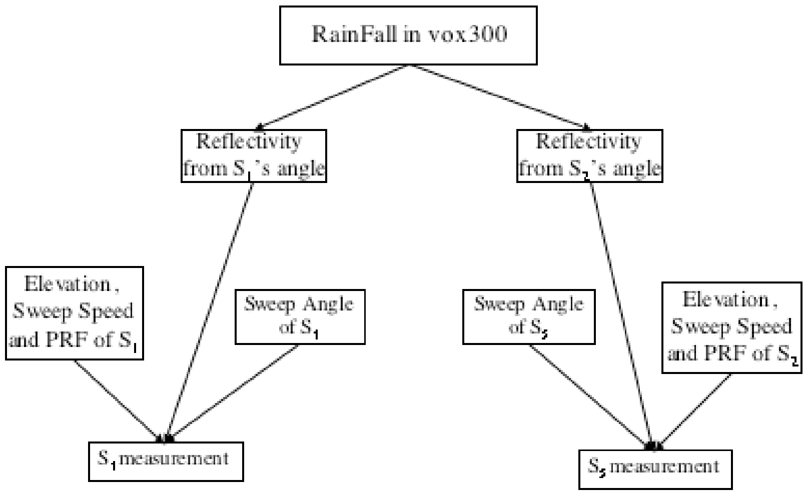

RainFall in vox300
Reflectivity from S1's angle
Reflectivity from S2's angle
Elevation, Sweep Speed and PRF of S1
Sweep Angle of S1
Sweep Angle of S2
Elevation, Sweep Speed and PRF of S2
S1 measurement
S2 measurement

Figure 3: Bayes Net for Rainfall with 2 Sensors

We will go through this calculation for the first clause of R1: {1: 0-36, R}: 11.46155205. Both the rainfall (E in Expression 1) and the sensor reading nodes (M in Expression 1) take on the values LOW, MED, and HIGH. For each value of E, there are 3 values for M ({M1=LOW} , {M1=MED} , and {M1=HIGH}), where M1 denotes the reflectivity measurement for M1. So the full equation is:

$P(R{=}LOW)\big[P(M1{=}LOW|R{=}LOW)Val(R{=}LOW, D(M1{=}LOW))$
$+ P(M1{=}MED|R{=}LOW)Val(R{=}LOW, D(M1{=}MED))$
$+ P(M1{=}HIGH|R{=}LOW)Val(R{=}LOW, D(M1{=}HIGH))$
$- Val(R{=}LOW, dec{=}MED)\big] +$

$P(R{=}MED)\big[P(M1{=}LOW|R{=}MED)Val(R{=}MED, D(M1{=}LOW))$
$+ P(M1{=}MED|R{=}MED)Val(R{=}MED, D(M1{=}MED))$
$+ P(M1{=}HIGH|R{=}MED)Val(R{=}MED, D(M1{=}HIGH))$
$- Val(R{=}MED, dec{=}MED)\big] +$

$P(R = HIGH)\big[P(M1 = LOW|R = HIGH)Val(R = HIGH, D(M1 = LOW))$
$+ P(M1{=}MED|R{=}HIGH)Val(R{=}HIGH, D(M1{=}MED))$
$+ P(M1{=}HIGH|R{=}HIGH)Val(R{=}HIGH, D(M1{=}HIGH))$

$- Val(R{=}HIGH, dec{=}MED)\big]$

$=$

$.33\big[.82 \cdot Val(R{=}LOW, dec{=}LOW)$
$+ .09 \cdot Val(R{=}LOW, dec{=}MED)$
$+ .09 \cdot Val(R{=}LOW, dec{=}HIGH)$
$- Val(R{=}LOW, dec{=}MED)\big] +$

$.33\big[.09 \cdot Val(R{=}MED, dec{=}LOW)$
$+ .82 \cdot Val(R{=}MED, dec{=}MED)$
$+ .09 \cdot Val(R{=}MED, dec{=}HIGH)$
$- Val(R{=}MED, dec{=}MED)\big] +$

$.33\big[.09 \cdot Val(R{=}HIGH, dec{=}LOW)$
$+ .09 \cdot Val(R{=}HIGH, dec{=}MED)$
$+ .82 \cdot Val(R{=}HIGH, dec{=}HIGH)$
$- Val(R{=}HIGH, dec{=}MED)\big]$

$=$

$.33\big[.82 \cdot 20$
$+ .09 \cdot 1$
$+ .09 \cdot 0$
$- 1\big] +$

$.33\big[.09 \cdot -10$
$+ .82 \cdot 20$
$+ .09 \cdot 1$
$- 20\big] +$

$.33\big[.09 \cdot -20$
$+ .09 \cdot -10$
$+ .82 \cdot 20$
$- (-10)\big]$

$=$

11

Where R is the amount of actual rainfall.

4.3 Configuration Selection

Since our *configurations* include both a start angle and an end angle, the size of our *task utility functions*' *explicit domains* could be very large. Even if we limit ourselves to nearby sensors and only consider *configurations* that sweep over the task, the combinatorics of the possible start and end angles is prohibitive. We thus introduce a preprocessing step which lists all the reasonable angles for each sensor to be sweeping and the reasonable setting sets (R, H, etc) for each of those angles. Then when we are evaluating *configurations* for a task, we only need to consider the combinations of settings that were decided on in the preprocessing step. What we mean by a reasonable angle is best explained in the context of an example. Recall the map shown in Figure 1. Our approach is, for each sensor, to consider only the angles that are needed to look at some set of tasks, and only the setting sets that make sense for those tasks. So, for sensor one, the only *configurations* we are interested in are {0-16, R}, {16-36, H}, {0-36, R}, {0-36, H}, and {0-36, RH}. Similarly, for sensor two, we are only interested in {200-217, R}, {180-200, H}, {180-217, R}, {180-217, H}, and {180-217, RH}. So when it comes time to generate the bids, we only need to consider pairs and singletons of these, reducing the number of bid clauses, for this toy example, from about $3 * 10^{11}$ (for a discretization of one) to 70.

4.4 Combinatorial Auction and Modifications

The BOB algorithm represents the bids as a graph, wherein each node is a bid clause and each edge represents mutual exclusivity between two clauses. The algorithm is then a depth first search through the space of bid combinations. It heuristically selects a clause to consider, removes that clause from the graph, and then finds the best solution with and without the clause. Much of the speed of the algorithm comes from the fact that removing clauses that have been considered will often decompose the graph into several smaller graphs, thus speeding up the calculation (since the algorithm is at the worst case exponential in the number of nodes). See [5] for a thorough presentation of this algorithm.

As was mentioned in section 2, we had to modify the combinatorial auction program to accommodate the multi-setting aspect of the problem. The modification we made was to change the conditions under which two clauses are mutually exclusive. In a normal combinatorial auction, two bid clauses are mutually exclusive if they are of the same bid or if they refer to some of the same items (resources). In our implementation, two clauses are mutually exclusive if they are in the same bid or if they refer to some of the same items AND their requested settings for those items differ. That is to say, if a large number of bids want S1 in the same *configuration*, then they can all be satisfied as far as S1 is concerned.

5 Results

First let us look at some examples of bid clauses to confirm that the task utility computation matches our intuitions. Consider a single rainfall task: for one sensor in rainfall setting, it will compute a utility of 11.46 if the sensor is sweeping across 30 degrees, 9.22 for 60 degrees, and 6.98 for 90 degrees. This is the behavior we want, since a larger the sweep angle means the radar spends less time looking at any given point. Also, if we look at multiple sensors for the same task, the utility computed will be 14.38 for three sensors sweeping 30 degrees in rainfall setting as opposed to the 11.46 for one. If we increase the amount of noise in the environment, the gap widens, with the task utility being 5.42 for one sensor and 8.02 for 3 (an increase of 47% as compared to 25% for the low noise case).

For the example we have been discussing, the total runtime is 18.5 seconds on a 1794.602 MHz Pentium 4 processor with 512MB of ram. 13.2 of these seconds are spent on Bayesian inference: a total of 1152 inferences are drawn in the calculation of 48 bid clauses. 42 of these clauses are sent to the auction algorithm (clauses with low values are pruned), which takes .047 seconds to solve them.

For comparison, if we add in a third task, the total runtime is 105 seconds. 86 of these seconds are spent on bayesian inference: a total 7464 inferences are drawn in the calculation of 280 bid clauses. 209 of these clauses are sent to the auction algorithm, which takes 1.3 seconds to solve them.

The results so far are encouraging in that they suggest that the combinatorial auction itself, the most daunting part of the problem from a combinatorial standpoint, appears to run fast enough (especially with additions that will be discussed in Section 6). This result alone shows that the approach is promising. Most of the computation in our implementation occurs during the evaluation of the *task utility functions* which could in principal be evaluated by any means we like. We will discuss ways to speed up both the *task utility function* evaluation and the rest of the system in Section 6

6 Summary and Future Work

In addressing the problems set forth by the CASA domain, we have formulated a new variant on the classic resource allocation problem, which we call the *setting-based resource allocation problem.* This problem reflects the challenges posed in the allocation of specialized sensor resources in the type of distributed sensor network in which sensors have multiple settings, each of which could be useful to multiple tasks. Further, we have proposed a solution to this problem that takes advantage of the locality of resources and tasks that is common to such domains.

This solution involves translating tasks into bids that can be solved by a modified combinatorial auction, thus allowing us to take advantage of recent developments in the solution of such auctions. We developed an information-theoretic procedure for accomplishing this translation which allows us to model the use of the sensors to a high degree of accuracy. Further, since this modeling is done with Bayesian networks, it is relatively simple to enter domain knowledge provided by experts.

We then implemented this approach for the CASA domain, allowing us to test the soundness of the approach and to confirm that the answers provided by our model make sense. Our implementation also gives us preliminary runtime data which helps us understand both the applicability of the approach to the domain and what features might be useful in speeding up the approach. We will discuss these features in the following paragraphs.

We are discussing a number of modifications to our system, designed to speed up both the auction and the *task utility function* evaluation. In fact, follow on work by a colleague has resulted in allowing us to solve the auction for three tasks and two sensors in under a millisecond. His change involves merging the bid clauses of multiple tasks for the same *configuration* into one clause-like structure. We are also considering taking advantage of the anytime nature of the BOB algorithm, so that, in an online setting, there is no risk of us not having any allocation when one is needed.

We are also discussing speeding up the *task utility function* evaluations in several ways. The least drastic change we are considering is using an approximate inference algorithm for Bayesian inference in place of the junction tree algorithm we are currently using. Another modification we are considering is compiling the Bayes nets to remove the middle-layer nodes (such as the reflectivity nodes in Figure 3). This can be done by accounting for the effects of the removed nodes in the conditional probability tables of the remaining nodes. A more significant enhancement we are considering is adding a case-base that would keep track frequently evaluated *configurations* and then interpolate based on the closest known configuration. Perhaps the most promising idea is to replace our Bayes nets with a neural network trained to mimic their output. If successful, either of these last two enhancements would give us similar evaluations to those provided by our current models, but in a fraction of the on-line time.

One final idea to speed up the system differs from those we have just discussed in that it promises to reduce the total number of *task utility function* evaluations that will have to be performed: We are considering the possibility of using a bid-eliciting auction such as the one described in [3]. This algorithm, instead of taking a set of bids as input, requests individual bid clauses (or relationships between the bids) as they are needed in the search. This could potentially reduce the number of *task utility function* evaluations performed.

APPENDIX A: Example Bids for R1 and H1

```
R1:
{1: 0-36, R }:  11.46155205
XOR
{2: 180-200, H 1: 0-36, R }:  11.461552049999998
XOR
{2: 180-217, RH 1: 0-36, R }:  11.461552049999998
XOR
{2: 180-217, R 1: 0-36, RH }:  11.461552050000002
XOR
{2: 200-217, R 1: 20-36, H }:  11.461552050000005
XOR
{2: 180-217, R 1: 0-36, R }:  10.677776554500001
XOR
{2: 180-217, R 1: 20-36, H }:  11.461552050000005
XOR
{1: 0-36, RH }:  9.447072195000002
XOR
{2: 180-217, H 1: 0-18, R }:  11.461552049999998
XOR
{2: 180-200, H 1: 0-36, RH }:  9.447072195000004
XOR
{2: 180-200, H 1: 0-18, R }:  11.461552049999998
XOR
{2: 180-217, RH 1: 20-36, H }:  9.447072195
XOR
{2: 180-217, H 1: 0-36, RH }:  9.447072195000004
XOR
{2: 200-217, R }:  11.46155205
XOR
{2: 180-217, RH 1: 0-36, H }:  9.447072195
XOR
{2: 180-217, R 1: 0-18, R }:  10.677776554500001
XOR
{2: 180-217, RH 1: 0-36, RH }:  8.565913173344997
XOR
{2: 200-217, R 1: 0-36, RH }:  11.461552050000002
XOR
{1: 0-18, R }:  11.46155205
```

```
XOR
{2: 180-217, R 1: 0-36, H }:  11.461552050000005
XOR
{2: 180-217, RH 1: 0-18, R }:  11.461552049999998
XOR
{2: 200-217, R 1: 0-36, R }:  10.677776554500001
XOR
{2: 180-217, R }:  11.46155205
XOR
{2: 200-217, R 1: 0-18, R }:  10.677776554500001
XOR
{2: 180-217, RH }:  9.447072195000002
XOR
{2: 200-217, R 1: 0-36, H }:  11.461552050000005
XOR
{2: 180-217, H 1: 0-36, R }:  11.461552049999998

H1:
{2: 200-217, R 1: 0-36, H }:  11.461552049999998
XOR
{2: 180-217, RH 1: 0-36, H }:  11.461552049999998
XOR
{2: 180-217, RH 1: 20-36, H }:  11.461552049999998
XOR
{2: 200-217, R 1: 20-36, H }:  11.461552049999998
XOR
{2: 180-217, RH 1: 0-18, R }:  9.447072195
XOR
{2: 180-217, H }:  11.46155205
XOR
{2: 180-217, H 1: 0-36, R }:  11.461552050000005
XOR
{2: 180-217, H 1: 20-36, H }:  10.677776554500001
XOR
{1: 0-36, RH }:  9.447072195000002
XOR
{1: 20-36, H }:  11.46155205
XOR
{2: 180-200, H 1: 20-36, H }:  10.677776554500001
XOR
```

```
{2: 180-217, H 1: 0-18, R }:  11.461552050000005
XOR
{2: 180-200, H 1: 0-36, R }:  11.461552050000005
XOR
{2: 200-217, R 1: 0-36, RH }:  9.447072195000004
XOR
{2: 180-217, RH 1: 0-36, RH }:  8.565913173344997
XOR
{2: 180-217, RH 1: 0-36, R }:  9.447072195
XOR
{2: 180-200, H 1: 0-36, H }:  10.677776554500001
XOR
{2: 180-217, R 1: 20-36, H }:  11.461552049999998
XOR
{2: 180-200, H 1: 0-36, RH }:  11.461552050000002
XOR
{2: 180-217, R 1: 0-36, RH }:  9.447072195000004
XOR
{2: 180-217, H 1: 0-36, H }:  10.677776554500001
XOR
{2: 180-200, H 1: 0-18, R }:  11.461552050000005
XOR
{2: 180-200, H }:  11.46155205
XOR
{2: 180-217, H 1: 0-36, RH }:  11.461552050000002
XOR
{2: 180-217, R 1: 0-36, H }:  11.461552049999998
XOR
{2: 180-217, RH }:  9.447072195000002
XOR
{1: 0-36, H }:  11.46155205
```

References

[1] A. Arnt and S. Zilberstein. Attribute measurement policies for cost-effective classification., 2004.

[2] Berthe Y. Choueiry, Boi Faltings, and Guevara Noubir. Abstraction Methods for Resource Allocation. In *Proceedings of the Workshop on*

Theory Reformulation and Abstraction, pages 2–71/2–90, Jackson Hole, Wyoming, 1994.

[3] W. Conen and T. Sandholm. Minimal preference elicitation in combinatorial auctions. In *IJCAI-2001 Workshop on Economic Agents, Models, and Mechanisms*, pages 71–80, 2001.

[4] S. Gorinsky and H. Vin. Additive increase appears inferior, 2000.

[5] Tuomas Sandholm and Subhash Suri. BOB: Improved winner determination in combinatorial auctions and generalizations.

[6] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *IJCAI*, pages 1102–1108, 2001.

[7] William E. Walsh and Michael P. Wellman. Efficiency and equilibrium in task allocation economies with hierarchical dependencies. In *IJCAI*, pages 520–526, 1999.

[8] M. Wellman, P. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves, and W. Walsh. A trading agent competition, 2000.