Applying Constraint Reasoning to Real-world Distributed Task Allocation

Paul Scerri, Pragnesh Jay Modi, Wei-Min Shen and Milind Tambe
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{scerri,modi,shen}@isi.edu, tambe@usc.edu

ABSTRACT

Distributed task allocation algorithms requires a set of agents to intelligently allocate their resources to a set of tasks. The problem is often complicated by the fact that resources may be limited, the set of tasks may not be exactly known, and the set of tasks may change over time. Previous resource allocation algorithms have not been able to handle over-constrained situations, the uncertainty in the environment and/or dynamics. In this paper, we present extensions to an algorithm for distributed constraint optimization, called *Adopt-SC* which allows it to be applied in such real-world domains. The approach relies on maintaining a probability distribution over tasks that are potentially present. The distribution is updated with both information from local sensors and information inferred from communication between agents. We present promising results with the approach on a distributed task allocation problem consisting of a set of stationary sensors that must track a moving target. The techniques proposed in this paper are evaluated on real hardware tracking real moving targets.

1. INTRODUCTION

A distributed task allocation problem requires a set of distributed agents to negotiate about how to allocate a set of resources to a set of tasks. Previous research has proposed distributed constraint satisfaction (DisCSP) as an approach to model and solve general multi-agent coordination problems [2] and recent work has applied the DisCSP model to distributed task allocation problems[3]. However, three key difficulties remain in unlocking the promise of distributed constraint reasoning in real-world multi-agent domains. First, many real-world domains require agents to reason about limited resources. Task allocation problems with limited resources are *over-constrained* problems[1]. Existing distributed constraint satisfaction representations simply return failure when no solution is possible. Instead, we desire agents to abandon the least important tasks, and thus find a solution that is "closest" to a satisfactory solution. Second, the set of tasks may not be completely known in advance and agents may face ambiguity in determining which tasks are present. Existing DisCSP methods require a known, completely specified, DisCSP problem as input and do not allow agents to deal with uncertainty during the problem solving process. Finally, the set of tasks may change over time. Tasks may appear/disappear and agents must monitor the environment and detect such changes. Current DisCSP techniques do not deal with such dynamism.

In this paper, we consider how to apply distributed constraint reasoning to domains where resources are limited, agents will have noisy information about which tasks are present and tasks change dynamically over time. We present techniques for addressing each issue. First, we generalize the DisCSP representation to a Distributed Constraint Optimization Problem (DCOP), which allow solutions to have degrees of quality or cost. We briefly present a general, optimal algorithm for DCOP named *Adopt-SC* [4]. The problem generalization and Adopt-SC algorithm allow agents to reason about allocating resources to only the most important tasks when resources are limited, given that a set of tasks is known and static. Second, agents maintain a probability that a particular task is present. The probability distribution is updated using both information from the agent's sensors and using information inferred from the distributed constraint problem-solving between agents. This probability distribution inherently captures the uncertainty that the agent has about the presence of tasks. This information informs Adopt-SC about the correct values to assign to variables. Third, the dynamics of the environment are handled by continually updating the probability distribution, informing Adopt-SC when "significant" changes occur.

The DCOP algorithm uses a representation with discrete variables and values. Tasks are mapped to variables and resources allocated a task are mapped to values. However, the underlying probabilistic reasoning uses a continuous probability distribution over tasks while the DCOP algorithm requires a discrete task status to decide which tasks to assign resources to (i.e., which variables require values). To achieve this, each probability is mapped to one of three task status: P, NP or U, representing present, not present and unknown respectively. While tasks with status P or NP can be handled by any DCOP algorithm tasks with status U require special treatment. In particular, when the agent must reply to another agent regarding resource allocation for a task it will assume that the other agent is correct in its assessment of the task as being present or absent. For example, suppose Agent 1 sends a message to Agent 2 allocating resources to Task 1. If Agent 2 has the status of Task 1 as U it will accept Agent 1's status of P (which it infers because Agent 1 assigns resources to the task) and reply accordingly. However, if Agent 2 has the status of the task being either P or NP it will reply using that status, regardless of what status Agent 1 had for the task.

Figure 1 shows the basic design of an agent, showing the

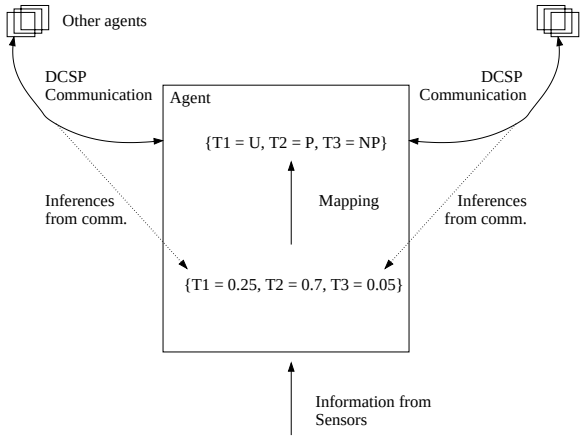

Figure 1: Diagram of the basic architecture. Arrows show the main channels of communication.

channels of communication and the information that flows along those channels. Notice that information flows in both directions, i.e., it flows down from the high level negotiation reasoning and up from the low level sensor readings. This allows the agent to take advantage of both local information, i.e., sensor readings, and global information, i.e., inferred information from other agents, giving it an accurate picture of which tasks are present.

The paper is structured as follows: Section 2 describes a describes the Distributed Constraint Optimization problem, a strict generalization of DisCSP and how it can be used to represent Distributed Resource Allocation. Section 3 then presents a new algorithm for DCOP that allows agents to find optimal solutions to distributed optimization problems. Section 4 presents the details of the probabilistic approach to handling uncertainty in a DCOP algorithm. Section 5 presents results using the algorithm on real hardware.

2. DCOP REPRESENTATION

The following section describes the Distributed Constraint Optimization (DCOP) problem and how we can map it into a resource allocation problem.

A Distributed Optimization Problem consists of n variables $V = \{x_1, x_2, ...x_n\}$, each assigned to an agent, where the values of the variables are taken from finite, discrete domains $D_1, D_2, ..., D_n$, respectively. Only the agent who is assigned a variable has control of its value and knowledge of its domain. For each pair of variables x_i, x_j, we are given a *cost function* $f_{ij} : D_i \times D_j \rightarrow N \cup \infty$. The cost functions are the analogue of constraints from DisCSP. The objective is to find a complete assignment $\mathcal{A}^*$ of values to variables such that the total cost is minimized. (An assignment is *complete* if all variables in V are assigned some value.) More formally, let $\mathcal{C} = \{\mathcal{A} \mid \mathcal{A}$ is a complete assignment of values to variables in $V\}$. We wish to find $\mathcal{A}^*$ such that $\mathcal{A}^* = \arg\min_{\mathcal{A} \in \mathcal{C}} F(\mathcal{A})$, where

$$F(\mathcal{A}) = \sum_{x_i, x_j \in V} f_{ij}(d_i, d_j) \quad , where\ x_i \leftarrow d_i, \\ x_j \leftarrow d_j\ in\ \mathcal{A}$$

This is a strict generalization of the standard representation of the Distributed Constraint Satisfaction Problem. For

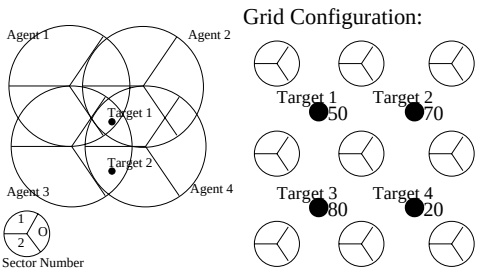

Figure 2: Sensor sector schematic(left) and a grid layout configuration with weighted targets (right)

example, a "hard" constraint is modeled as having infinite cost for all pairs of variable values that violate the constraint and zero cost otherwise.

We formulate distribute resource allocation as a DCOP in the following way. Formally, we have a set of all possible tasks T_a, where $|T_a| = K$. N tasks will be present at any time, $N < K$. The set of tasks actually present are $T_p(|T_p| = N)$. For each task in T_a, a DCOP variable has a value from $\{Allocated, NotPresent, Ignore\}$, representing present, not present and ignore respectively. Resources must be allocated to all tasks with P value. If a task is assigned the value I, a cost function w: $T_a \rightarrow N \cup \infty$ quantifies the cost of ignoring the task. The DCOP requires agents to choose value for variables such that resources are assigned to only the most important tasks and ignore tasks with small costs when resources are limited. Section 3 describes Adopt-SC, an algorithm for optimally solving DCOP problems.

2.1 Sensor Network Domain

A concrete instantiation of a resource allocation problem is the following distributed sensor network domain[6]. It consists of multiple fixed sensors, each controlled by an autonomous agent, and multiple targets moving through their sensing range. Each sensor is equipped with three radar heads, each covering 120 degrees. Resource contention may occur because an agent may activate at most one radar head, or sector, at a given time. Hence, if different tasks require that different sectors are used then both tasks cannot be performed simultaneously. Three sensors must turn on overlapping sectors to accurately track a target. For example in Figure 2 (left), when agent 1 detects a target in its sector 0 it must coordinate with neighboring agents so that they activate their respective sectors that overlap with agent 1's sector 0. Targets in a particular region are *tasks* that need to be completed/tracked and a choice of three sensors to track a target. Figure 2(right) shows a configuration of 9 agents and an example of resource contention. Since at least three neighboring agents are required to track each target and no agent can track more than one, only two of the four targets can be tracked. The agents must find an allocation that minimizes the weight of the ignored targets.

3. ALGORITHM FOR DCOP

We briefly describe the *Adopt-SC* algorithm for DCOP [4]. Adopt-SC (Adopt with Save Context) is a version of the Adopt algorithm which saves all search paths in order to increase efficiency[5]. The Adopt-SC algorithm requires variables to have a fixed total priority order. Any ordering is

sufficient and lexicographic ordering is the simplest method. Figure 4.a shows constraints pointing from higher priority agents to lower ones. Two agents x_i, x_j are *neighbors* if their cost function f_{ij} is not a constant. Given the constraint graph and priority ordering, agents form a search tree where each agent has at most one parent and there are no neighbors in different subtrees. Two agents x_i, x_j are *linked* if they are neighbors, if x_i is the parent or child of x_j, or if they are both linked to a common descendent. The solid arrows in Figure 4.b show links. The priority ordering and tree can be formed in a preprocessing step, or alternatively, can be discovered during algorithm execution. For simplicity of description of the algorithm, we will assume the tree is already formed in a preprocessing step. We will use the term *parent* to refer to an agent's immediate higher priority agent in the tree, *children* to refer to an agent's immediate lower priority agents in the tree, and *linked descendents (ancestors)* to refer to linked agents lower (higher) in the tree. In Figure 4.b, x_1 is the parent of x_2, x_2 is the parent of x_3 and x_3 is the parent of x_4 (as defined by the transfer of VIEW messages, which will describe later). $x4$ is a linked descendent of $x2$.

A set of variable/value pairs specifying a (possibly incomplete) assignment is called a *view*).

- **Definition:** A *view* is a set of pairs of the form $\{(x_i,d_i), (x_j, d_j)...\}$. A variable can appear in a view no more than once. Two views are *compatible* if they do not disagree on any variable assignment and a view is *larger* than another if it contains more variables.

The deficiency of a value of a variable in a view is determined by the sum of its cost functions.

- **Definition:** The *local deficiency* of a given view vw wrt variable x_i is defined as

$$\delta(x_i, vw) = \sum_{x_j \in V} f_{ij}(d_i, d_j) \quad , where\ x_i \leftarrow d_i,\ x_j \leftarrow d_j\ in\ vw$$

Procedures from the Adopt algorithm are shown in Figure 3. x_i represents the agent's local variable and d_i represents its current value. The algorithm begins by each agent instantiating its variable concurrently and sending this value to all its connected lower priority agents via a VALUE message. After this, agents asynchronously wait for and respond to incoming messages. Lower priority agents choose values that have the least deficiency given the current values of higher priority agents stored in the $Currentvw$ variable. This often leads to quick, possibly suboptimal solutions. In order to escape local minima, lower priority agents report feedback to higher priority agents. When a lower priority agent evaluates its local cost functions and realizes the system is incurring cost greater than *tolerance level* τ, shown in Line (iii) of Figure 3, it constructs a VIEW message which contains its current view of the higher priority agents' assignments and the associated amount of cost. It sends this VIEW message only to the lowest higher priority connected agent (its parent). As an agent receives VIEW messages, it maintains the set of views and associated costs reported to it from its children. Then, the agent will either abandon its current variable value in favor of one with less total cost (Line (ii)) or pass a VIEW message up to *its* lowest higher priority agent. Figure 4.b shows the flow of VALUE and VIEW messages between agents as the algorithm executes. Important properties of this algorithm are that an agent only

 # Currentvw: Current view of linked ancestor's variable values
 # d_i: x_i's current variable value
Initialize: $Currentvw \leftarrow \{\}; d_i \leftarrow$ null;
  $\forall x_l \in Children$:
    $c(x_l, \{\}) \leftarrow 0$;
    $Views(x_l) \leftarrow \{\}$;
  hill_climb;
when received (**VALUE**, (x_j, d_j))
  add (x_j, d_j) to $Currentvw$;
  hill_climb;
when received (**VIEW**, x_l, vw, $cost$)
  add vw to $Views(x_l)$;
  $c(x_l, vw) \leftarrow cost$;
  hill_climb;
procedure hill_climb
  $\forall d \in D_i$:
    $e(d) \leftarrow \delta(x_i, Currentvw \cup \{(x_i, d)\})$;
    $\forall x_l \in Children$:
      $vw \leftarrow$ largest view in $Views(x_l)$ compatible with $Currentvw \cup \{(x_i, d)\})$;
      $e(d) \leftarrow e(d) + c(x_l, vw)$;
  choose d that minimizes $e(d)$; — **(ii)**
  if $d_i \neq d$ then
    $d_i \leftarrow d$;
    SEND (**VALUE**, (x_i, d_i)) to all linked descendents;
  end if;
  choose d_h where $(parent, d_h) \in Currentvw$;
  if $e(d_i)$ greater than τ then — **(iii)**
    SEND (**VIEW**, x_i, $Currentvw$, $e(d_i)$) to parent;

Figure 3: Procedures from the Adopt-SC algorithm

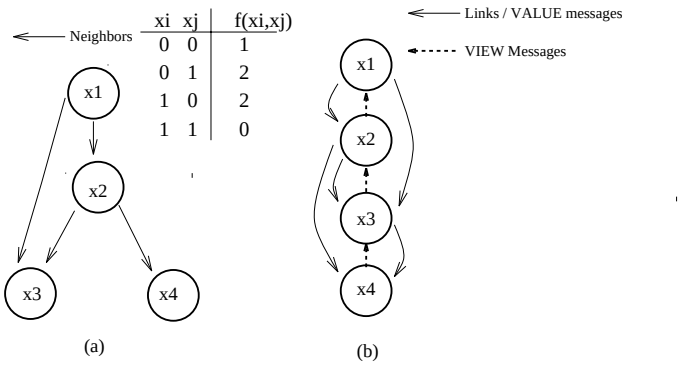

Figure 4: (a) Constraint graph and associated priority ordering. (b) Flow of VALUE and VIEW messages between agents.

stores variable/value information about connected variables, rather than building a complete path from root to leaf, and agents are able to search for solutions asynchronously.

When time is limited, we can increase the value of the τ parameter, which in turn allows agents to ignore smaller costs by not reporting to higher priority agents. Although this gives up optimality gurantees, it prevents higher priority agents from switching their values, thus allowing the system to reach a stable state more quickly. A key property of this local tolerance is that agents still attempt to find values that minimize costs, as long as it can be done locally without backtracking. In general, the τ parameter would need to be engineered using domain knowledge but our point is that Adopt is "tune-able" in this way.

4. DEALING WITH UNCERTAINTY AND DYNAMICS IN ADOPT-SC

In this section we lay out the details of the use of an underlying probability distribution for Adopt-SC and its integration into the workings of the resource allocation algorithm.

For each task in T_a, a task status from $\{P, NP, U\}$, representing present, not present and unknown respectively, is maintained. The set of task status for $T_a$is called the *task status vector*, denoted $V^{A_n}$for Agent An. The task status is used by Adopt-SC to decide which variable values to assign. Tasks, which map to variables, with status P could be assigned either value $Allocated$ or $Ignored$. Tasks with status NP must be assigned the value $NotPresent$. Finally, tasks with status U can be assigned any value, depending on the value at other agents. A *task status vector projection* is the set of tasks in $V^{A_n}$where the value is of a certain type, e.g. the set $V_U^{A_n}$is those tasks for which A_n has the value U. The aim of the uncertainty reasoning is to make $V_P^{A_n} = T_p$, i.e., to make the tasks the agent thinks are present be the same as the tasks that are actually present.

For each $T \in alltasks$ the probability that the task is currently present is $Pr(T)$. The agent maintains this probability for each task, i.e., it maintains probabilities $\mathbf{PR} = \{Pr(T_1) \ldots Pr(T_N)\}$. Each agent's probability distribution is maintained locally, hence agents may have different views on which tasks are present. A function maps $\mathbf{PR}$ to V^{A_n}. The details of this mapping are somewhat arbitrary and need to chosen in a domain dependent manner. In the sensor network domain we use the following mapping:

$$\begin{aligned} \text{if } Pr(T) < 0.2 \quad & \text{then } NP \\ \text{else if } Pr(T) > 0.8 \quad & \text{then } P \\ \text{else} \quad & U \end{aligned} \tag{1}$$

4.1 Algorithm

The low level probabilistic component and Adopt-SC's task allocation algorithm run asynchronously. Earlier, we described the workings of the task allocation algorithm, in this section we describe the algorithms operating on the probabilistic representation and describe how they are integrated with Adopt-SC.

The interface between the probabilistic representation and Adopt-SC is intentionally kept simple so that changes to either component do not require significant changes in the other. Information flows from the probabilistic representation to Adopt-SC via messages indicating that the status of a task has changed. For example, a message is sent when the status of a task changes from U to P. In the other direction, Adopt-SC sends messages indicating which tasks other agents believe to be present, whenever it receives a communication providing that information. For example, if Agent 2 communicates the presence of Task 1 to Agent 1, Adopt-SC at Agent 1 will send a message to the probabilistic component indicating the presence of Task 1. Notice that Adopt-SC sends this message regardless of its beliefs about the task, since the probabilistic component can use the information to change its probability distribution, reinforcing or lessening the probability a task is present.

The probabilistic component has two distinct modes of operation. Which mode of operation to use is determined by whether Adopt-SC has allocated the agent to a task that is currently present. In the sensor network domain this can be determined by whether Adopt-SC has specified using a radar head that can detect a target. If Adopt-SC has allocated resources of the agent to a present task then the probabilistic component has a passive monitoring role. If Adopt-SC allocates the agent to a task that is not present then the probabilistic component acts pro-actively to determine which tasks are present. In other words, the probabilistic component is only pro-active when Adopt-SC is failing. Adopt-SC is given responsibility for making decisions if it can because it is able to make globally optimal allocation decisions while the probabilistic component can only randomly choose between detected targets. When Adopt-SC is failing, i.e., it is asking for actions towards a task that is not present, the probabilistic component, with a superior ability to determine which tasks are present, assumes control.

In its passive, monitoring mode the probabilistic component updates its probability distribution and informs Adopt-SC when the status of a task changes. In its active mode, the probabilistic component takes actions which aim to determine the presence of a task as quickly as possible. Determining what action will most readily determine the presence of tasks can be easily inferred from its sensor model. That is, the action that is most likely to resolve uncertainty, given the current state of $\mathbf{PR}$ is chosen.

In the following, each of the different observations are discussed in some detail. In the next section, results using the approach with real hardware are presented.

4.2 Using Observations to Reduce Uncertainty

The probability distribution over tasks and its mapping to V^{A_n} is at the heart of the approach presented here. Hence, maintaining as accurate as possible distribution is essential to the success of the approach. Creating and maintaining this distribution in a noisy, dynamic world requires both combining multiple measurements to reduce uncertainty while giving more weight to the most recent measurements to ensure the current situation is captured. Four pieces of information are used to update the probability distribution.

- Updates based on observations made while performing a task, using a learned environment model. Formally, this information is $Pr(T|S)$, where S is a sensor reading.
- Updates made based on inferences from overheard communications from other nodes. Formally, this information is $Pr(T|M)$ where M is a message.

- Updates made based on knowledge of the dynamics of the domain. In particular, the probability that a task is present given the probability that it was present earlier. Formally, this information is $Pr(T_t|Pr(T_{t-1}))$, where $Pr(T_t)$ is the probability task T is present at time t.
- Updates based on probabilistic information about relationships between tasks. Formally, this information is $Pr(T_1|Pr(T_2) \wedge \ldots \wedge Pr(T_N))$.

We refer to each type of information as an observation, denoted O. Each of the types of observation provides some evidence about the presence of a task, T. In particular, given a model of the types of observation that can be received we can calculate $Pr(T|O)$. That evidence should be combined with previous evidence to make $Pr(T)$ more accurate. However, since the situation changes dynamically, more recent evidence should be weighted more heavily than older information. The integration of the new observations with the previous evidence uses a variation on the standard probability rule:

$$Pr(T|O) = \frac{Pr(O|T) \times Pr(T)}{Pr(O)}$$

In this equation O is the new observation. $Pr(O|T)$ is the probability of getting the observation given that the task is present. This probability is calculated in different ways, depending on the type of observation. For example, a model of the sensors provides this information for sensor observations. $Pr(T)$ is the a priori probability of task T. Since we know the probability that the task was present in the previous time step we can use that information to calculate the probability that the task is present in the current time step. That is:

$$Pr(T_t) = \frac{Pr(T_t|T_{t-1}) \times Pr(T_{t-1})}{Pr(T_{t-1}|T_t)}$$

where $Pr(T_t)$ is the probability of task T being present at time t. For simplicity, we set $\frac{Pr(T_t|T_{t-1})}{Pr(T_{t-1}|T_t)} = w$. Essentially, this assumes that the dynamics of the environment are uniform across tasks and times. Thus, the calculation of the probability of a task given a new measurement and an previous probability is:

$$Pr(T_t|O) = \frac{Pr(O|T_t) \times wPr(T_{t-1})}{Pr(O)}$$

(Our current implementation uses a simplified version of this calculation.) The integration of new observations iteratively updates **PR**. When any $Pr(T)$changes enough that it causes the status of a task to change a message is sent to Adopt-SC which then may start a new round of negotiation to determine a new optimal task allocation.

4.2.1 Updates from Sensors

In the sensor network domain, essentially the same actions are taken to detect tasks as to perform those tasks. Using a learned model of the environment the agent can leverage measurements taken in the course of performing a task to reason about the presence of all tasks. This technique for reducing uncertainty is purely local, i.e., the agent uses only local information to reduce uncertainty. Table 1, shows part of the model for a particular sensor in a particular configuration of the sensor network domain. Column one gives the sector in which the agent takes the reading. Column two gives the strength of the reading, higher numbers indicate stronger readings which in turn indicate there is a target close by. Columns 3-6 give the percentage of times the task is present when a reading of that strength is taken in that sector. Weak readings do not give accurate information about the presence of tasks. For example, if a reading of strength 0 is taken in sector 0 very little information is garnered about which tasks are likely to be present. Strong readings provide much more information. For example, if a reading of strength 6 is taken in sector 0 that reading was due the presence of either Task 4 or 5. Even with this strong reading the agent is not able to completely determine which task is present and must rely on information from other agents to determine exactly which task it is.

		Tasks					
Sector	Reading	T1	T2	T3	T4	T5	T6
0	0	10	20	14	14	18	22
0	1	3	21	17	7	21	28
0	2	9	13	10	21	24	20
0	3	8	17	9	20	27	16
0	4	0	0	17	22	27	31
0	5	0	0	13	25	47	13
0	6	0	0	0	42	57	0
1	0	9	21	21	13	18	15
1	1	7	30	19	7	23	11
1	2	24	28	11	8	9	18
1	3	24	29	20	2	3	18
1	4	0	16	24	20	30	8
1	5	0	3	26	40	26	2
1	6	0	0	64	35	0	0
2	0	33	17	14	8	10	16
2	1	27	18	18	13	9	13
2	2	17	11	16	31	14	7
2	3	19	39	21	4	9	4
2	4	32	29	9	0	18	9
2	5	42	47	1	0	8	0
2	6	0	100	0	0	0	0

Table 1: Mapping from sensor readings to probability distribution across tasks.

For the sensor network domain the model is created by running the system for several hours, comparing the readings of the sensors with the actual state of the world. This is effectively a simple form of supervised learning.

4.2.2 Updates from Overheard Communication

In order to find a good (optimal, if time is available) allocation of resources to tasks, Adopt-SC requires agents negotiate as described above. Since the task status vector for each agent will be different, agents can infer useful information from communications from other agents. The local sensing actions of each agent are suited to detecting particular tasks. Inferring information from communication allows an agent to leverage the ability of another agent to accurately detect a particular task.

At the level of Adopt-SC negotiations the agents are not

dealing with the probability a task exists, instead they are using *Ignore*, *Allocated* and *NotPresent*. Messages with *Ignore* or *Allocated* imply the presence of the task, while messages with *NotPresent* imply the absence of a task. Since each agent uses the same probabilistic reasoning if an agent sends a message indicating the presence (absence) of a task it must have a probability above (below) its threshold. If the thresholds are reasonably high (low) communicated messages provide good information about the presence of tasks.

However, a communication does not give detailed information about the certainty with which the communicating agent believes in the presence of the task. For example, if agent A sends a message allocating resources to a task, T, the agent receiving the message can only infer $threshold < Pr_A(T) < 1.0$. Potentially, the agents could also communicate their perspective of the probability that a task is present but this would add to the required communication bandwidth and has so far not been required.

4.2.3 Scheduling Observation Actions

Adopt-SC assigns weights to tasks, prioritizing tasks with higher weights. Normally, if the status of some task is U the agent will not actively try to allocate resources to that task, nor will it take actions to determine whether or not the task is actually present. However, if it is currently allocating resources to a task with lower weight than a task with status U it will periodically schedule actions to resolve the uncertainty surrounding that task. In particular, in the sensor network domain it can switch to the sector most likely to determine whether or not the task is present. This behavior ensures that important tasks are not ignored simply because no agent checks whether the task is present. However, neither do the agents spend time checking for tasks that are of lower priority than the one to which they are currently allocating resources.

4.3 Modeling dynamics

The final method for updating the probability distribution is to update based on probabilistic, temporal relationships between tasks. For example, in the sensor network domain since tasks are related to target locations future tasks can be probabilistically predicted based on the limitations of the movement of targets. Specifically, if a target is in a particular area, making a particular task present when that task is no longer present the target should be in an adjacent area, hence providing information about which tasks might be present next.

5. HARDWARE EXPERIMENTS

The probability approach was inspired by work with real hardware. In particular, experience working with the hardware showed that handling dynamism and uncertainty are important aspects of tasks allocation in the real world. In this section, we describe the hardware setup, experiment and results.

The sensors were arranged in a diamond in a small room inside the Information Sciences Institute (a very noisy environment for the sensors). The configuration is shown in Figure 7 (a photograph of the radar and target (toy train) are shown in Figure 6). The lines on the sensors show the orientation of the radar heads. Notice that the sensors at the ends of the room did not need to change sectors, while the

Figure 5: Left: A Doppler radar for tracking moving targets. Right: Target to be tracked.

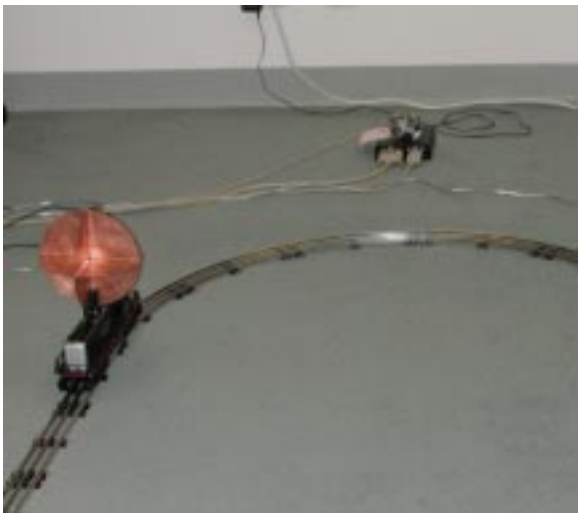

Figure 6: Photograph of target (train) with sensor during experiment.

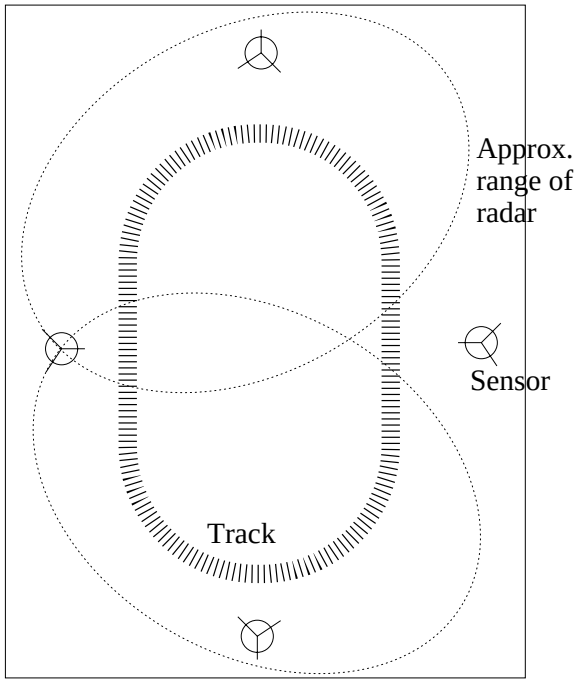

Figure 7: The configuration of the room, sensors and target track for hardware experiments. Dotted ellipses from sensor on left hand sensor show approximate range of two of the sensors radar heads.

sensors on the sides of the room needed to switch between two sectors.

The hardware experiments were performed before the probability model was fully implemented[1]. Adopt-SC interpreted raw sensor readings directly as either indicating the presence or absence of a task (i.e., rather than going via the probability model). If the agent does not activate the sensor that would determine the presence of a task in a specific amount of time the status of that tasks switches to U. As with the algorithm described above, if the task allocation algorithm suggests a sector where there is currently a task it pursues that task. When the task allocation algorithm suggests a sector where there is no target detected it switches to looking in a sector marked U. Finally, information received from other agents about the presence of tasks is taken at face value and its task status updated according. Hence, if another agent sends a message indicating the presence of a task that the agent has marked as U the agent will change the status of the task to present when the message is received.

While the approach used on the hardware is superficially quite different to the probabilistic model presented above, the idea of integrating several different sources of information to reduce uncertainty is common. The probabilistic approach was formalized based on analysis of results of this simpler approach.

The aim of the sensor network is to obtain an accurate track of one or more moving targets. Creating such a track involves a variety of algorithms working together, e.g., the task allocation algorithm and an algorithm for combining

[1]The full model was not implemented until after we lost access to the hardware.

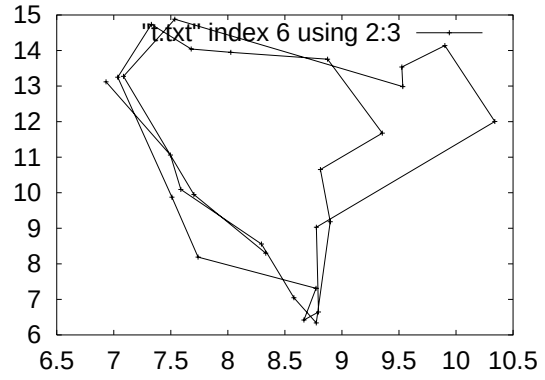

Figure 8: Track produced by four sensors following a target moving on an oval track.

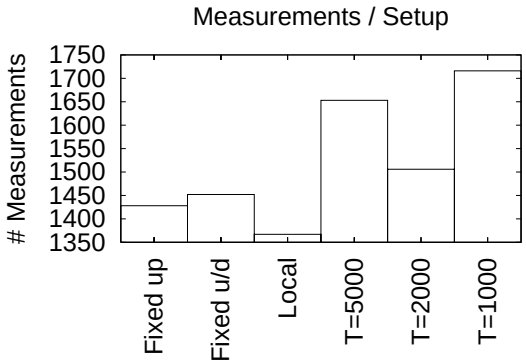

Figure 9: Number of measurements made by various algorithms.

measurements from multiple sensors. A sample track is shown in Figure 8. For this experiment we are only interested in the performance of the task allocation algorithm and in particular the way the algorithm deals with uncertainty. Hence, the quality of the track produced is not a good metric, rather we use a metric which gives the number of measurements of the target taken by the agents. The more measurements taken the more often the sensors were focused on the target and not searching for it or looking in the wrong sector. Three different algorithms were used. The results are shown in Figure 9 (the x-axis shows the number of measurements taken and the y-axis shows the algorithm used). The first algorithm used a fixed configuration of sectors based on the known track of the target. One configuration had the sensors on the sides of the room both looking towards one end of the room ("fixed up" in the figure), while the other had the sensors on the sides of the room looking to opposite ends of the room ("fixed u/d" in the figure). The next algorithm ("local" in the figure) used only local sensing information, changing sectors whenever it failed to sense a target in the sector it was currently using. Finally, Adopt-SC was used with various timeout lengths (1 second – "T=1000" in the figure, 2 seconds – "T=2000" and 5 seconds – "T=5000"). Each algorithm was run three times, each time for 20 minutes. The values shown on the graphs are the average number of measurements across the three runs.

Adopt-SC performed clearly better than the other algorithms because the four nodes together were better able to resolve uncertainty and find the target than the localized algorithms. The "local" algorithm performed worst because it was most susceptible to the noise in the environment. A single false reading indicating the presence of a target would result in the agent wasting a significant amount of time. The algorithms utilizing information from others as well as their own information were less susceptible to single noisy measurements. The reason for the difference in performance of Adopt-SC with different time out values is not exactly clear but it likely related to the speed of the moving target.

6. SOFTWARE EXPERIMENTS

The full probability model and associated algorithms have been implemented for use in a simulator of the sensor network domain. Systematic experiments are currently being performed. Initial results appear promising with agents accurately finding targets despite a large amount of simulated noise.

7. CONCLUSIONS

Using DisCSP for task allocation in the real world involves dealing with three issues that are not addressed by DisCSP algorithms. In particular, over-constrained situations, dynamics and uncertainty need to be addressed. In this paper we have proposed extensions to an asynchronous, distributed constraint optimization algorithm that addresses these issues. In particular, we use a probability model over possible tasks, updating that model with information from sensors, communication from other agents and knowledge of the dynamics of the environment. Reasoning based on that probability model was used to choose actions for not only which tasks to attend to but also to choose actions to find whether tasks are currently present. The approach was tested on both real hardware and in a simulator and was shown to perform well. Future work will involve using more details of the underlying probability distribution in the constraint satisfaction algorithm.

8. REFERENCES

[1] K. Hirayama and M. Yokoo. An approach to over-constrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In *Proc. of the 4th Intl. Conf. on Multi-Agent Systems(ICMAS)*, July 2000.

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

[3] P. J. Modi, H. Jung, W. Shen, M. Tambe, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. In *Proc of Constraint Programming*, 2001.

[4] P. J. Modi, W. Shen, and M. Tambe. Distributed constraint optimization and its application. Technical Report ISI-TR-509, University of Southern California/Information Sciences Institute, 2002.

[5] P. J. Modi, W. Shen, and M. Tambe. Distributed constraint optimization and its application. In *Proceedings of AAAI'02 Workshop on Coalition Formation*, 2002.

[6] BAE Systems / Sanders. Ecm challenge problem. http://www.sanders.com/ants/ecm.htm, 2001.

[7] Noam M. Shazeer, Michael L. Littman, and Greg A. Keim. Solving crossword puzzles as probabilistic constraint satisfaction. In *AAAI/IAAI*, pages 156–162, 1999.