Are multiagent algorithms relevant for real hardware? A case study of distributed constraint algorithms

ABSTRACT

Researchers building multi-agent algorithms typically work with problems abstracted away from real applications. The abstracted problem instances allow systematic and detailed investigations of new algorithms. However, a key question is how to apply algorithm, developed on an abstract problem, in a real application. In this paper, we report on what was required to apply a particular distributed resource allocation algorithm developed for an abstract coordination problem in a real hardware application. A probabilistic representation of resources and tasks was used to deal with uncertainty and dynamics and local reasoning was used to deal with delays in the distributed resource allocation algorithm. The probabilistic representation and local reasoning enabled the use of the multi-agent algorithm which, in turn, improved the overall performance of the system.

1. INTRODUCTION

In the field of multiagent systems, researchers often abstract out key real-world coordination problems for investigation within software testbeds[2]. Abstracting out such key problem features is critical to enable a systematic investigation of the coordination issues, unhindered by other complex factors in real-world environments. One such coordination problem that has received significant attention recently is distributed resource allocation[9]. Researchers have provided distributed constraint representation and reasoning, market-based approaches[10], argumentation[6], and several other approaches to address such problems. Significant success has been reported in development of new and efficient algorithms.

The key question addressed in this paper is understanding techniques and principles required to apply such coordination algorithms, developed in the context of software testbeds, to real-world environments. While some real-world environments are essentially software or simulation tools which may more easily match the assumptions in such coordination algorithms, other real-world environments involve physical hardware, where uncertainty, real-time constraints and dynamism prevails. It is the latter environment that provides the bigger challenge in building applications, and they are the focus of this paper. Identifying the principles to apply multia-gent algorithms to such domains will help in two key ways. First, it will help future researchers in developing such applications, potentially expanding the range of applications for multiagent techniques. Second, it may identify key weaknesses of existing algorithms that make such applications difficult, enabling multiagent researchers to identify assumptions in their research that may need to be modified to ease such applications.

A concrete example that is the vehicle of our investigation is applying distributed constraint reasoning (DCR) algorithms to distributed resource allocation problems, in particular, in a real-world distributed sensor domain. DCR algorithms have been extensively investigated in the multiagent literature [3], but typically in the context of abstract domains such as graph coloring problems, and have not been tested previously in the context of real-world uncertain domains. The distributed sensor network domain investigated in this paper exhibits three important real-world features which have not been previously modelled in abstract DCR domains: a) task uncertainty – the set of tasks to be performed can only be known probabilistically, b) real-time constraints – resources must be allocated and tasks must be performed within hard time deadlines and c) task dynamism – the set of tasks are changing over time. Novel techniques are required to apply existing DCR techniques to real-world domains that exhibit these properties.

In this paper, we consider how to apply DCR to domains where agents will have noisy information about which tasks are present, resources are limited and tasks change dynamically over time. We present techniques for addressing each issue and how these techniques can be incorporated into an existing DCR algorithm with minimal effort. We use a general, optimal algorithm for DCR named *Adopt-SC* to perform distributed resource allocation. Adopt-SC is suitable for distributed resource allocation problems because it allows agents to *optimize*, i.e., reason about allocating limited resources to only the most important tasks. The key idea in this work is to apply the Adopt-SC algorithm to a real-world domain using a two-layered architecture. The general principle is to allow a coordination algorithm such as Adopt-SC to effectively perform the functions for which it was designed by encapsulating the domain details into a lower layer. The lower layer is a probabilistic component that deals with task uncertainty and dynamics and allows an agent to do local reasoning when time for coordination is not available. Adopt-SC runs as the higher layer coordinating inter-agent activities. The lower layer provides information to the higher layer about which tasks are present, leaving Adopt-SC free to operate as if tasks are known. The higher layer, which is able to communicate with other agents, provides non-local information to the lower layer thereby allowing it to update its probability model about which tasks are present.

The two-layer architecture deals with the three "domain details"

described above as follows. First, since agents cannot determine with certainty which tasks must be performed, they must represent the 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. The dynamics of the environment are handled by continually updating the probability distribution, informing Adopt-SC when "significant" changes occur. Adopt-SC allocates resources to tasks with a probability of being present above some threshold. Second, the lower layer provides an ability to make fast, local resource allocation decisions when time for coordination is limited. Finally, while Adopt-SC has a capability for finding suboptimal solutions quickly, it takes some time to find even the first solution once the situation changes. To ensure reasonable operation in a real-time environment, Adopt-SC is augmented by "locally intelligent reasoning" which can perform a simple allocation of resources while Adopt-SC works on finding a good global allocation of resources.

We present results obtained from an implementation of the system described above on real hardware. The results show that the two-layered architecture is effective at tracking real moving targets. The probability model and local reasoning enabled the coordination algorithm to deal with the difficulties posed by a real-world distributed sensor domain. The use of a two-layered architecture, where domain details are hidden from the coordination algorithm, allowed us to incorporate an existing "off the shelf" distributed constraint reasoning algorithm into a real application. As far as we are aware, this paper reports on the results of the first successful application of distributed constraint reasoning on real hardware. We believe this is a significant first step towards moving multiagent algorithms developed on abstract problems out of the lab and into the real world.

2. SENSOR NETWORK DOMAIN

The distributed sensing application investigated in this paper consists of multiple fixed sensors, each controlled by an autonomous agent, and multiple targets moving through their sensing range[7]. Each sensor is equipped with three radar heads, each covering 120
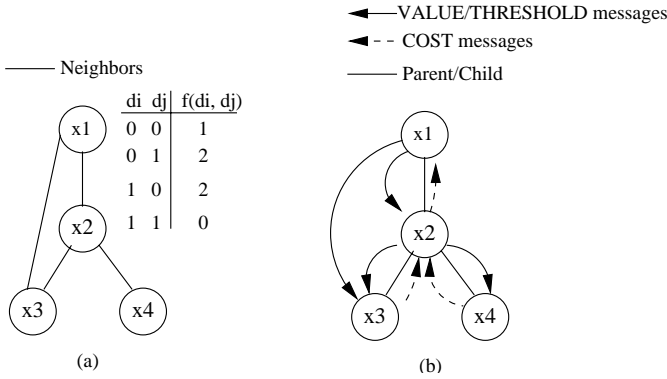
Figure 2: (a) Constraint graph. (b) Communication graph.

resources are assigned to only the most important tasks and ignore tasks with small costs when resources are limited. Section 3.1 describes Adopt-SC, an algorithm for optimally solving DCOP problems.

3.1 Adopt-SC

The Adopt algorithm [5] performs a distributed asynchronous search for the optimal solution using linear space at each agent. Adopt has been theoretically shown to be complete, i.e., the globally optimal solution is guaranteed to be found using only localized, asynchronous communication. Communication is local in the sense that agents only communicate with neighboring agents. Adopt-SC (Adopt with Save Context) is a version of the Adopt algorithm that stores all partially explored search paths in order to increase efficiency, but at the expense of memory [4]. Adopt-SC requires exponential space at each agent in the worst case. However, in practice for the sensor network domain discussed in this paper, we found that the benefits of the efficiency outweighted the cost of the additional memory requirement.

We briefly describe the Adopt algorithm. Agents are ordered in a Depth-First Search (DFS) tree and each agent chooses its variable value concurrently. The DFS tree defines parent and child relationships. Figure 2.b shows a DFS tree formed from the constraint graph in Figure 2.a – x_1 is the root, x_1 is the parent of x_2 and x_2 is the parent of both x_3 and x_4. Each agent sends its value choice (via a VALUE message) to the descendents in the DFS tree with which it has a constraint. Each agent then computes a *bound interval* on cost for its subtree given these value assignments and asynchronously reports this information (via a COST message) to its parent in the DFS tree. A bound interval consists of a least lower bound and a least upper bound on solution cost. The size of the bound interval decreases over time as cost information percolates up the tree to an agent from its children. Figure 2.b shows the flow of messages between agents. The asynchronous message passing of variable values from parents to descendents and bound intervals (COST messages) from children to parent continues until each agent's least lower bound is equal to its least upper bound, in which case the globally optimal solution has been found. This is guaranteed to eventually occur and the algorithm will terminate with the optimal solution. In addition, Adopt can take as input a cost threshold, which is used for terminating with a sub-optimal solution when there is insufficient time to find the optimal solution.

4. APPLYING ADOPT-SC IN A DISTRIBUTED SENSOR DOMAIN

To use Adopt-SC in a distributed sensor domain it was necessary to add an extra component which maintains a probabilistic representation of the currently present tasks. This component abstracts away the details of the domain for Adopt-SC and allows it to find good solutions for a simplified problem. The probabilistic component also performs resource allocation itself, when Adopt-SC does not have solutions immediately available. In the remainder of this section, we describe the workings of this probabilistic component and its integration with Adopt-SC.

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. Notice, that the probabilistic component does this reasoning asynchronously and in parallel to Adopt-SC's resource allocation reasoning.

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.

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.

4.1 Probabilistic Task Representation

Figure 3 shows the channels of communication and the information that flows along those channels in an agent. 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.

For each task in T_a, a task status from $\{P, NP, U\}$, representing present, not present and unknown respectively, is maintained. The
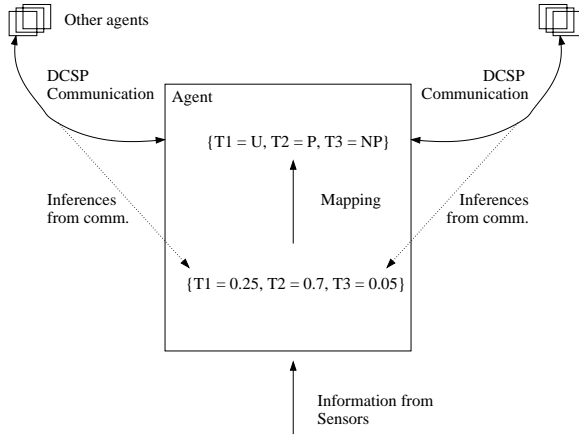
Figure 3: Diagram of the basic information flows around an agent.

set of task status for T_a is called the *task status vector*, denoted V^{A_n} for Agent An. The task status vector 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 $Ignored$ by Adopt-SC. Tasks with status NP must be assigned the value $NotPresent$ by Adopt-SC. Finally, tasks with status U can be assigned any value, depending on the value at other agents by Adopt-SC. 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 $T \in T_a$ 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 different agents may have different probabilities that a task is 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:

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

Maintaining as accurate as possible distribution is essential to the success of the approach. To create and maintain this distribution in a noisy, dynamic environment requires the combination of multiple measurements to reduce uncertainty, 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 Bayes' 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:

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

The integration of new observations iteratively updates $\mathbf{PR}$. When any $Pr(T)$ changes enough that it causes the status of a task to change, e.g., NP to P, a message is sent to Adopt-SC which then may start a new round of negotiation to determine a new optimal task allocation.

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, agents do not spend time checking for tasks that are of lower priority than the one to which they are currently allocating resources.

4.1.1 Updates from Sensors

In the sensor network domain, the same domain actions are taken to detect tasks as to perform those tasks. Using a learned model

Reading	Task							
	T1	T2	T3	T4	T5	T6	T7	T8
0	0.83	0.69	0.65	0.51	0.39	0.39	0.78	0.89
1	0.16	0.29	0.28	0.27	0.19	0.15	0.15	0.10
2	0.0	0.01	0.05	0.17	0.17	0.14	0.03	0.0
3	0.0	0.0	0.0	0.03	0.15	0.11	0.02	0.0
4	0.0	0.0	0.0	0.0	0.05	0.05	0.0	0.0
5	0.0	0.0	0.0	0.0	0.01	0.13	0.0	0.0

Table 1: Probability of a particular radar sector getting a particular reading when a task is present.

Strength	0	1	2	3	4	5
Probability	0.83	0.09	0.02	0.01	0.01	0.02

Table 2: Probability of getting a reading of a certain strength from a particular sensor and sector.

Figure 4: Left: A Doppler radar for tracking moving targets. Right: Target to be tracked.

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 $Pr(O|T)$ for a subset of zones for a particular node and sector. Column 1 gives the strength of the reading, i.e., the observation, with higher numbers representing stronger readings. Readings of strength 0 and 1 cannot be distinguished from noise. Columns 2-9 show the probability of getting a reading of that strength given that the task is currently present. In this example, the sensor can give little information about the presence of Task 1, since even if the task is present the readings will be no stronger than noise. On the other hand, for Task 6, the sensor will get a reading of strength 5 13% of the time. Table 2 shows the a priori probability of getting readings of various strengths. The table shows that readings of strength 5 are quite rare. Using Bayes' rule and the initial probability of Task 6 being present, a reading of strength 5 would increase the probability of Task 6 being present. For the distributed sensor domain, these probabilities tables can be calculated analytically from a model of the sensor.

4.1.2 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.

4.1.3 Dynamics and Task Relationships

Finally, the probability distribution is updated using a model of the dynamics of the domain and a model of relationships between tasks. The dynamics model captures probabilistic temporal relationships between tasks. For example, in the distributed sensor domain, we model the probability that Task 2 is present at $t = 2$ if Task 1 is present at $t = 1$. Since, we are tracking trains moving on set tracks this model can be quite accurate, but for the purposes of the experiments below we simply model that the object is likely to move to an adjacent position in the next time step. The model of relationships between tasks is useful when there are multiple tasks that have some probabilistic relationship between them. For example, in the distributed sensor domain for a military application, detecting an object at one point can lead to expectations about where other objects might be because objects might be moving in some sort of formation. Such a model can further reduce uncertainty and, thus, improve the information that Adopt-SC uses to allocate resources.

5. HARDWARE EXPERIMENTS

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 6 (a photograph of the radar and target (toy train) are shown in Figure 5). 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 sensors on the sides of the room needed to switch between two sectors.

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 measurements from multiple sensors. A sample track is shown in Figure 7. For this experiment we are only interested in the performance of the task allocation algorithm and in particular the way the algorithm deals with un-
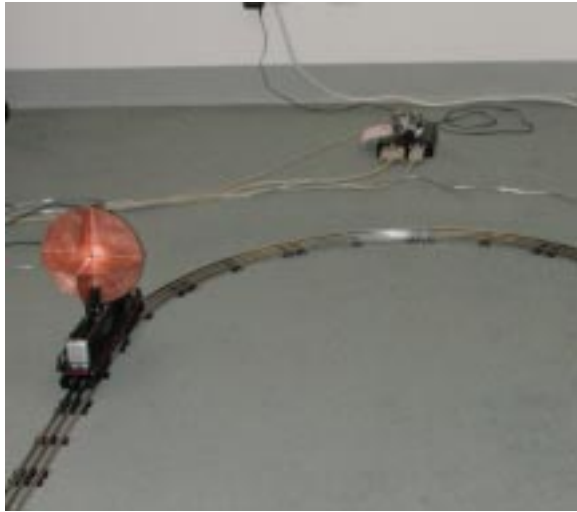
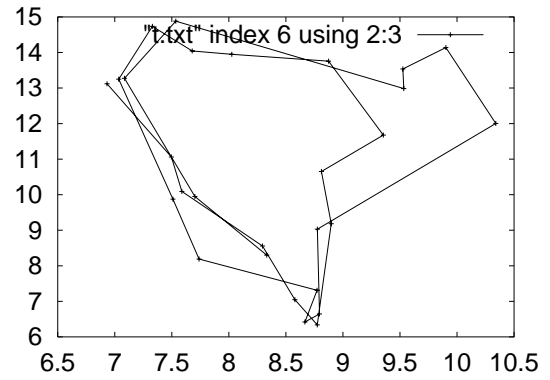**Figure 5: Photograph of target (train) with sensor during experiment.**

Figure 7: Track produced by four sensors following a target moving on an oval track.
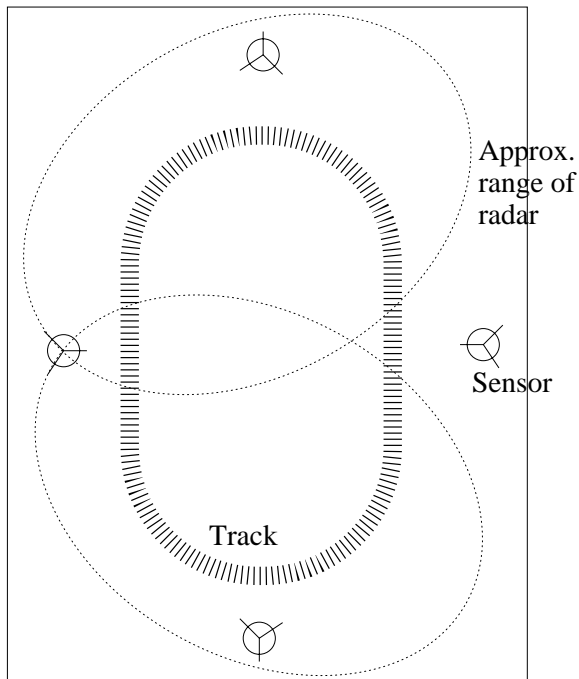

Figure 6: The configuration of the room, sensors and target track for hardware experiments. Dotted ellipses from sensor on left hand sensor show approximate range of two of the sensors radar heads.
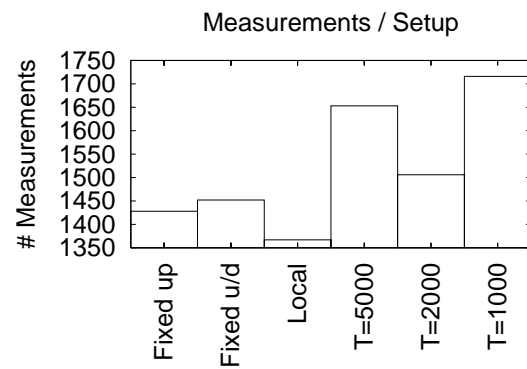

Figure 8: Number of measurements made by various algorithms.

certainty. 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 8 (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. CONCLUSIONS AND FUTURE WORK

Using a multiagent coordination for task allocation in a real world application involves dealing with issues that are not addressed in an algorithm developed on an abstract problem. In particular, dynamics, uncertainty and real-time constraints 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. Future work will use a realistic simulator of the problem to investigate in detail the factors that effect performance. Specifically, we intend to investigate the effect of the speed at which the set of present tasks changes on the usefulness of the algorithm.

7. REFERENCES

[1] Collin and Dechter. A distributed solution to the network consistency problem. *AI Journal*, pages 242–251, 1990.

[2] Steve Hanks, Martha Pollack, and Paul Cohen. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. *AI Magazine*, 14(4):17–42, 1993.

[3] 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.

[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, M. Tambe, and M. Yokoo. An asynchronous complete method for general distributed constraint optimization. In *Proc of Autonomous Agents and Multi-Agent Systems Workshop on Distributed Constraint Reasoning*, 2002.

[6] S. Parsons, C. Sierra, and N.R. Jennings. Agents that negotiate by arguing. *Journal of Logic and Computation*, 1998.

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

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

[9] O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *Proceedings of IJCAI'95*, pages 655–661, 1995.

[10] William Walsh and Michael Wellman. A market protocol for decentralized task allocation. In *Proceedings of the International conference on multi-agent systems*, pages 325–332, Paris, July 1998.

[11] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.