

AFRL-IF-RS-TR-2004-14
Final Technical Report
January 2004



INCREMENTAL NEGOTIATION AND COALITION FORMATION FOR RESOURCE-BOUNDED AGENTS

SRI International

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J130

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-14 has been reviewed and is approved for publication.

APPROVED:

/s/
DANIEL E. DASKIEWICH
Project Engineer

FOR THE DIRECTOR:

/s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE JANUARY 2004	3. REPORT TYPE AND DATES COVERED FINAL Aug 99 – May 03	
4. TITLE AND SUBTITLE INCREMENTAL NEGOTIATION AND COALITION FORMATION FOR RESOURCE-BOUNDED AGENTS			5. FUNDING NUMBERS C - F30602-99-C-0169 PE - 62301E PR - H358 TA - 01 WU - 01	
6. AUTHOR(S) Charles L. Ortiz, Regis Vincent, Eric Hsu, Bruno Dutertre, Barbara Grosz, Timothy Rauenbusch, Sarit Kraus, Osher Yadgar				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Ave Menlo Park CA 94025			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/IFTB 3701 North Fairfax Drive 525 Brooks Road Arlington VA 22203-1714 Rome NY 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2004-14	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Daniel E. Daskiewich/IFTB/(315) 330-7731 Daniel.Daskiewich@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT <i>APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.</i>			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 Words) This report describes SRI's contributions toward the solution of the problem of real-time distributed resource allocation. For the most part, the distributed multi-sensor challenge problem utilized by the Defense Advanced Research Projects Agency's Autonomous Negotiating Teams (ANTs) program was used to motivate the research. However, the contributions are not restricted to the sensor domain. We model the resource allocation problem as a multiagent problem in which each resource is modeled as an agent which can communicate with other agents to exchange requirements or task information. Agent interactions generally take the form of message exchanges to support auction-style algorithms in which a mediator requests bids on a task or a collection of tasks and then receives bids from agents. Each bid encapsulates local information, important to the allocation decision, in the form of utility or cost estimates.				
14. SUBJECT TERMS Software Agents, Resource Allocation, Autonomous Negotiation			15. NUMBER OF PAGES 116	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Abstract

In this report we describe both algorithms and experimental results for what we refer to as *center-based* approaches to realtime distributed resource allocation. Resources are modeled as agents that communicate with each other to exchange resource requirements and task information; one or more distinguished “center agents” mediate task assignments and interactions between agents. Agents are also architected to monitor and contribute to team commitments. Several variations of center-based allocation are explored: (1) Dynamic Mediation is an approach which supports negotiations between agents in the context of a changing problem and resource situation and in which tasks can interact in both positive and negative ways; (2) Allocation Improvement addresses the problem of combinatorial resource allocation; and (3) the Distributed Dispatcher Manager (DDM) supports the hierarchical organization and management of massive agent-based systems, on the order of thousands of agents and tasks. We also present a solution to the problem of communications management in the context of a multisensor allocation problem.

Contents

1	Project overview	1
1.1	The problem	1
1.2	Major elements of our approach	4
1.3	Outline of report	5
2	Center-based negotiation	6
2.1	Introduction	6
2.2	Center-based task assignment	8
2.3	Negotiation in context	13
2.3.1	Allocation Improvement	15
2.3.2	Experimental Evaluation	16
2.4	Combinatorial task allocation	19
2.4.1	Incremental Task Allocation Improvement Algorithm	21
2.4.2	Empirical Evaluation	23
2.5	Dynamic negotiation	25
2.5.1	Rich bids	27
2.5.2	Experimental results and evaluation	32
2.6	System architecture: interleaving negotiation and execution	36
2.6.1	Visualization tools and geometric reasoning	39
2.6.2	Experimental results	41
2.6.3	Auction results	41
2.6.4	Mediation experiments	45
2.7	Summary and related work	50
3	Resource allocation in very large-scale environments	54
3.1	The Distributed Dispatcher Manager	54
3.2	The large scale ANTS challenge problem and the DDM	56
3.3	Descriptions of algorithms	58

3.3.1	The raw data transformation and capsule generation algorithm	61
3.3.2	Leader localInfo generation algorithm	66
3.3.3	The movement of a sampler agent	72
3.4	Simulation, experiments and results	74
3.4.1	Simulation environment	74
3.4.2	Evaluation methods	74
3.4.3	Results	78
3.4.4	Small Scale Results	85
3.5	Related work	89
3.6	Conclusions	92
4	Multi-channel communications scheduling	93
4.1	Scheduling Access to Radio Channels	94
4.1.1	Model	94
4.1.2	Special Case: Broadcast	95
4.1.3	Algorithms and Experiments	96
4.2	A Channel Reservation Protocol	97
4.3	Paper on Dynamic Scan Scheduling	99
5	Summary and conclusions	100
	Bibliography	103

List of Figures

1.1	Multi-sensor tracking.	3
2.1	Center-based Assignment Algorithm	10
2.2	Overview of CBA algorithms	11
2.3	Multisensor tracking.	12
2.4	Mediation Algorithm	14
2.5	Sketch of the Allocation Improvement Update Procedure	16
2.6	Comparison of Mediation algorithms in the 4-agent sensor domain	18
2.7	Comparison of mediation algorithms in the 20-agent sensor domain	19
2.8	Task interaction graph and cost function for Agent 1	20
2.9	Anytime Algorithm for Task Allocation	22
2.10	Improvement of task allocation cost under ITAI.	24
2.11	Dynamic Mediation Algorithm	29
2.12	Update procedure for Dynamic Mediation.	29
2.13	Example Construction of Probability Distribution	31
2.14	Results for static mediation.	34
2.15	Results for dynamic mediation.	35
2.16	Results for many task appearances.	36
2.17	Architecture	38
2.18	The UMass visualization tool	38
2.19	The geometry tracker in action.	40
2.20	Identifying bad tracker estimates.	41
2.21	Opportunism	42
2.22	Auctioning, 16 nodes, .5 feet per second, 240-second run.	42
2.23	Auctioning, 24 nodes, .5 feet per second, 240-second run.	43
2.24	RMS by projection length.	44
2.25	Messages by projection length.	44
2.26	Track quality: 0 second projections.	46
2.27	Track quality: 5 second projections.	46

2.28	Track quality: 10 second projections.	47
2.29	Track quality: 15 second projections.	47
2.30	Track quality by number of nodes.	49
2.31	Auctioning versus mediation: quality.	49
2.32	Auctioning versus mediation: messages.	50
3.1	DDM hierarchy information flow diagram.	59
3.2	Target sampling by one Doppler.	62
3.3	Finding a value of θ_0	64
3.4	The rawDataTransformation function	65
3.5	Capsule generation algorithm.	65
3.6	Obtaining new information algorithm.	68
3.7	mergeFunctions algorithm.	69
3.8	Example of a set of unusedCapsules received by the Finding_new_paths algorithm.	70
3.9	An example of an outcome of phase 1	71
3.10	Finding new paths algorithm.	73
3.11	Patrol movement pattern.	74
3.12	Simulating 2 Doppler radars tracking 30 targets. The dots rep- resent sampled target states. The shades of lines represent 100% and 50% tracked targets.	75
3.13	Simulating 20 Doppler radars tracking 30 targets. The dots repre- sent sampled target states and the lines represent tracked targets.	76
3.14	Tracking percentage by time in zone (Sec.).	78
3.15	Time to track distribution (Sec.).	79
3.16	Tracking duration distribution (Sec.).	80
3.17	Accurate tracked target percentage as a function of the number of levels.	80
3.18	Accurate tracking time (Sec.) as a function of the number of levels.	81
3.19	Maximum agent process time (Sec.) as a function of the number of levels.	82
3.20	Bytes transferred as a function of number of levels.	83
3.21	Average number of bytes received by a single agent as a function of the number of levels.	84
3.22	Accurate tracked target percentage as a function of dysfunctional samplers.	84
3.23	Accurate tracked target percentage as a function of dysfunctional first level leaders.	85

3.24	Accurate tracked target percentage of patrol as a function of lost communication messages between samplers and leaders.	86
3.25	Target tracking percentage and average time by the settings.	86
3.26	Target detection percentage and average time as function of the communication noise.	90
3.27	Tracking percentage and average time as a function of the number of Dopplers.	90
5.1	Meeting the desiderata: note that this table highlights the major elements (left-hand column) addressed by each approach (top row) examined in this report. The column “combinatorial allocation” refers to the algorithm described in Chapter 2.	102

List of Tables

4.1	Experiments on a 10×10 Sensor Grid	97
-----	---	----

Chapter 1

Project overview

This report describes work performed by SRI International, Harvard University, and Bar-Ilan University under Contract F30602-99-C-0169 for the DARPA Autonomous Negotiating Teams (ANTS) Program. At Harvard and Bar-Ilan Universities, part of this work was also funded under NSF grant number IIS9907482. Sections of Chapter 2 and Chapter 3 have appeared in the volume published in 2003 by Kluwer Academic Publishing entitled *Distributed Sensor Networks* [Lesser et al 2003]. The authors of those chapters herein acknowledge that copyrighted material; any reproduction of this report should respect that copyright.

1.1 The problem

In this report, we describe our contributions toward the solution of the problem of realtime distributed resource allocation. We use, for the most part, the distributed multi-sensor challenge problem proposed by the ANTS program to motivate our research. However, our contributions are not restricted to the sensor domain. We also describe the application of our ideas to other domains when we see that as pedagogically helpful, while at the same time drawing connections to problems in the sensor domain.

We model the resource allocation problem as a multiagent problem in which each resource is modeled as an agent which can communicate with other agents to exchange resource requirements or task information. Agent interactions generally take the form of message exchanges to support auction-style algorithms in which a mediator requests bids on a task or a collection of tasks and then receives bids from agents. Each bid encapsulates local information, important to the allocation

decision, in the form of utility or cost estimates.

Our focus has been on problem settings and solutions with the following characteristics:

Distributed: Any resource allocation algorithm should be distributed in the sense that it should not depend on some centralized repository of global information where allocation decisions must be made. Such an assumption would be overly restrictive given the constraints imposed within real world settings in which inter-agent communications may be limited.

Incremental and realtime: The time-stressed nature of realworld problem domains precludes the possibility of computing optimal resource allocations before execution. Instead, agents should negotiate partial, good-enough allocations which can later be refined if time permits.

Flexible task allocation: Task allocation mechanisms should be flexible in the sense that potential allocations can be explored either sequentially, in terms of possible task-resource pairs, or combinatorially in the form of sets of multiple tasks and resources. In the latter case, mechanisms should be able to deal with tasks that can interact: that is, in which the cost of doing several tasks is not simply the sum of the individual costs of each task.

Adaptive resource allocation: Dynamic problem settings in which new tasks or resources can appear (or disappear) during the allocation process require that it not be necessary for allocation processes to be re-started from scratch each time the global situation changes.

Adaptive communications: Since bandwidth communications are assumed to vary, allocation algorithms should be able to adapt to limits imposed by the communications medium.

Fault tolerant: Any solution should be fault tolerant in the sense that it should be adaptable to resource loss during execution, as opposed to requiring that the allocation be re-started from scratch.

Scalable: Any solution should be scalable to very large agent and task settings.

Figure 1.1 is an example of realtime resource allocation involving multi-sensor tracking. The figure shows an array of 9 doppler sensors. Each sensor has three sectors associated with it, labeled $\{1, 2, 3\}$. A sensor can turn on a sector and take

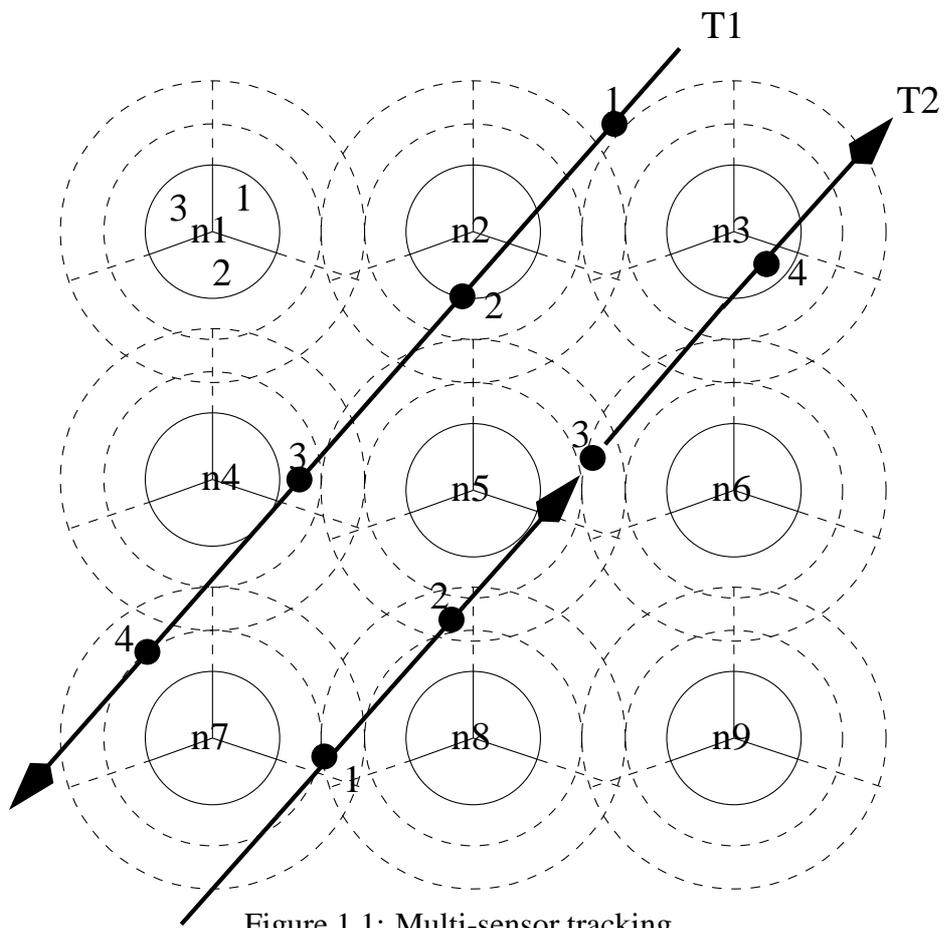


Figure 1.1: Multi-sensor tracking.

both frequency and amplitude measurements in order to determine velocity and distance. The more sectors that are on, the greater the power usage. The farther away the target is from the sensor, the lower the quality of the measurement. At least two sensors are necessary for estimating the location of an object; three sensors are desirable for obtaining a good quality estimate. Sectors require a 2 second warm up time and two objects which appear in the same sector and at the same time cannot be discriminated.

We conducted experiments on both a physical sensor suite and a simulation, called RadSim, based on the sensor suite. RadSim is a simulation environment developed by the Air Force Research Laboratory to support what will be referred to as the ANTS distributed resource allocation challenge problem. RadSim provides a simulated environment containing moving targets that are to be tracked (i.e. their positions determined over time) using simulated sensor nodes. Each sensor node is modeled as a 3-head Doppler radar unit and an 8-channel communications transceiver. The sensors are controlled by external agents that allows them to set sensor parameters, take measurements, and communicate with other agents (For details, see Chapter 2 of [Lesser et al 2003]). Due to the partial state of the ANTS program tracking software at the time of completion of this project, this report focuses on results from the simulation.

1.2 Major elements of our approach

Our approach to the multi-sensor tracking problem involves three stages: (1) initial coalition formation; (2) formation of a future coalition based on a projected object path; and (3) refinement of an existing coalition. We use the term coalition to refer to a group of agents who are joining together to perform some task; the members of a coalition can change as circumstances change. The initial coalition formation is a very quick process which assigns a group of three agents to a target. The initial coalition's task is to determine the position, direction and velocity of the target. In the second stage, one of the agents in the initial coalition takes that information and projects the path of the target into the future and then runs an auction or, as we will refer to its variants, a center-based algorithm on some set of agents that neighbor the projected path. The projected path is represented by a cone of uncertainty: the farther into the future, the greater the uncertainty in the projected position. In the final stage, a coalition can be adapted to changes that might occur in the path of a target: when a particular coalition, *A*, notices the change, it informs the remaining agents in the original coalition of that change so

that they can drop their commitments; A then runs a new auction to allocate new resources to the new path.

Given a projected track, $\{(11,t1),(12,t2),(13,t3), (14,t4)\}$, (for example, corresponding to the points 1, 2, 3, and 4 in track T1 shown in the figure) of location-time points, we have studied four allocation schemes: (1) a standard auction in which each point is auctioned to some subset of nodes; (2) a combinatorial allocation scheme in which sets of location-time points are considered simultaneously; (3) mediation, in which a mediator suggests allocations to some subset of nodes at each round of negotiation; and (4) a variation in which a mediator allocates agents to tasks through a hierarchical team organization.

1.3 Outline of report

The remainder of this document is organized in the following way. In Chapter 2 we explore variations of center-based algorithms for resource allocation; in particular, we focus on a variant which we call Dynamic Mediation that directly addresses the dynamic and realtime aspects of distributed resource allocation. In that chapter, we also present a combinatorial allocation algorithm as well as an agent architecture that supports monitoring of team commitments. In Chapter 3 we present a system called the Distributed Dispatcher Manager (DDM) for managing massive agent-based systems, on the order of thousands of agents and tasks. DDM is the first system able to manage such large collections; it does so by organizing agents hierarchically into teams. In Chapter 4 we present a solution to the problem of communications management in the context of the ANTS challenge problem. Finally, Chapter 5 summarizes our contributions.

Chapter 2

Center-based negotiation

2.1 Introduction

This chapter introduces a class of negotiation protocols appropriate for problem domains in which tasks can interact arbitrarily and in which agents must negotiate over the assignment of resources to tasks in dynamically changing settings. We use the term *negotiation* to simply refer to any distributed process through which agents can agree on an efficient apportionment of tasks among themselves. Our work has compelled us to reject various assumptions commonly made in the negotiation literature: these include assumptions that (1) the context in which a negotiation or bid is made is irrelevant to the negotiation and, consequently, that task costs or utilities can be assumed to be additive: we allow instead for the possibility of positive and negative task interactions; (2) the environment remains static during a negotiation: we allow for the possibility that important changes can occur during a negotiation that can affect the results of that negotiation; and (3) all changes in the environment can be anticipated during negotiation: in any realistic domain, the world may change in unexpected ways at execution time, requiring that a solution be adapted to those changes. We restrict ourselves to problems involving resource-bounded agents comprising cooperative teams. The requirement for resource-boundedness means that negotiation protocols must be temporally constrained in some way and that an optimal allocation will not always be possible. The restriction to cooperative teams corresponds to a restriction to agents that can be assumed to be honest and non-competitive.

The focus on non-additive domains is an admission that tasks can interact in interesting and significant ways and that such interactions should be kept in mind

during a negotiation. Therefore a decision to allocate a task to a particular agent cannot necessarily be made independently of a task allocation to another (or even the same) agent. For example, in the sensor challenge problem, if a deactivated emitter is activated, the beam is unstable and will not give reliable measurements for 2 seconds; therefore, if one task is immediately followed by another in the same sector, the beam will not require the 2 second warmup; this corresponds to a positive task interaction. As another example, consider that only one of the three detectors on a sensor can be scanned at a given time and each scan takes between 0.6-1.8 seconds; therefore, two sequential tasks that are less than 0.6 seconds apart and occur in separate sectors, will interact negatively. Consider another domain, such as a delivery domain. When negotiating whether two tasks can be taken on by the same agent (for example, a delivery to two different locations) agents must consider whether those tasks interact in such a way that the joint cost of performing both tasks might be greater or less than performing each separately (for example, in the former case, if the delivery points are in opposite directions entailing separate trips).

The rejection of the assumption that the world remains static during a negotiation requires that one develop negotiation protocols that allow a negotiation to be adapted to such changes, rather than requiring that the negotiation be re-started. The rejection of the assumption that the world also remain static during execution requires that agents negotiating be architected to monitor the management of deployed resources and flexibly respond to unexpected changes in a way that respects team-centered concerns: that is, an agent's commitments should not be limited to those resulting just from the negotiation but should extend dynamically to the team.

We shall, for the most part, use the distributed sensor domain to motivate our design decisions; this has also served as an experimental testbed for validation. However, our contributions toward the solution of these problems is not restricted to the sensor domain. Consequently, we will describe the application of our ideas to related domains when we see that as pedagogically helpful, while at the same time drawing connections to problems in the sensor domain.

Our focus in this chapter is on one important class of negotiation algorithms which we will refer to as *center-based* algorithms. Examples include sequential auctions, combinatorial auctions, and contract nets. When the objects over which agents negotiate correspond to tasks rather than physical objects (as in auctions), contract nets are equivalent to auctions. Hence, we shall view them interchangeably. At the heart of such mechanisms is a *center agent* (for example, an auctioneer or contractor) that collects bids on proposed allocations. Each bid is meant to

compactly encapsulate important local information (such as utility information) that can subsequently be made use of by the center in its decision on the best allocation. A center-based approach is very different from approaches in which some central coordinator has access to up-to-date information regarding the local states of agents and then uses that knowledge to compute optimal allocations. Furthermore, the amount of information contained in agents' local states may be large, thus rendering centralization infeasible, particularly in cases of communication delays and system faults.

Combinatorial auctions have been suggested as a promising method for exploring allocation of items that interact: agents have the freedom to choose particular bundles of items. However, as we shall see, they leave unanswered the question of how best to choose the bundles on which to bid and also assume that an agent's value does not interact or depend on the bids of other agents. Within the broad notion of a center-based negotiation mechanism, many variations are possible; those variations are not restricted to auctions. In fact, we will present one variation, which we call *mediation*, that usefully allows considerations about the context in which a bid is made.

2.2 Center-based task assignment

We define a center-based task assignment problem in the following way. We describe a system in terms of a set of possible “runs” or system executions, in the form of state transitions for each agent, including a distinguished run representing the actual execution. Formally, we have:

Definition 2.2.1 *Let \mathfrak{R} stand for the set of real numbers. A task allocation system, M , is represented as a five-tuple, $M = \langle A, T, u, \mathbf{P} \rangle$, where,*

1. $A = \{a_1, \dots, a_n\}$ is a set of n agents with some agent designated as the mediator,
2. $T = \{t_1, \dots, t_m\}$ is a set of m tasks,
3. $u : A \times 2^T \rightarrow \mathfrak{R} \cup \{\infty\}$ is a value function that returns the value which an agent associates with a particular subset of tasks,
4. An assignment, \mathbf{P} , or partition, of size n on the set of tasks T such that $\mathbf{P} = \langle P_1, P_2, \dots, P_n \rangle$, where P_j contains the set of items assigned to agent a_j .

We refer to each element of \mathbf{P} as a *proposal*. We will sometimes represent a proposal as an ordered m -tuple of agents, the i -th element of which corresponds to the agent assigned to perform task i . The special proposal \emptyset will represent the null

proposal, which corresponds to the situation in which no assignment of tasks is possible. For example, $P_5 = \langle a_1, a_5, a_3, a_1 \rangle$ corresponds to the allocation in which t_1 is assigned to agent a_1 , t_2 to agent a_5 , t_3 to agent a_3 , and so on. The level of detail encoded by each proposal is domain dependent. The algorithms described in this paper assume that each proposal encodes sufficient detail to enable each agent to evaluate its cost function.

The value function can represent, for example, the utility or cost attached to a task, or perhaps some multi-objective utility function. In the experiments described in this chapter, we associate either cost or utility to the value function, depending on the particular focus of the experiment. Where the value function ranges over a set of tasks consisting of only one element, we use the same notation and refer only to the single element; we assume throughout that this secondary use is clear from the context. There is some globally defined objective function, f , that determines the desirability of an assignment based on the value that each agent ascribes to the items to which it is assigned.

We assume the group objective is social welfare maximization and thus the objective function is given by:

$$f(p, A) = \sum_{a \in A} u(a, p)$$

where $p \in \mathbf{P}$.¹

The *negotiation problem* is that of choosing an element p^* of \mathbf{P} that maximizes the objective function:

$$p^* = \arg \max_{p \in \mathbf{P}} f(p, A).$$

The proposal chosen is called the *outcome* of the negotiation.

The amount of time available may or may not be known to the agents in advance. For instance, agents may need to end their negotiation and begin acting in a given number of milliseconds, or they may need to begin acting when some event occurs at an unspecified time in the future.

Both Mediation and combinatorial auctions are examples of algorithms that can be used to solve the assignment problem. They are both members of a class of algorithms we will call center-based assignment (CBA) algorithms. In a CBA

¹A common alternative objective is Pareto efficiency where the objective function can be defined as:

$$f(p, A) = \begin{cases} 0 & \text{if } \exists q \in \mathbf{P} \text{ such that } \forall a \in A, u(a, q) > u(a, p) \\ 1 & \text{otherwise} \end{cases}$$

```

let  $H \leftarrow \emptyset$ 
loop
  construct an announcement  $a(H)$ 
  send  $a(H)$  to each agent
  incorporate  $a(H)$  into  $H$ 
  receive “bid”  $b_j(a, H)$  from each agent
  incorporate  $b_j(a, H)$  into  $H$ 
  compute  $P(H)$ 
  UNTIL  $terminate(H)$  or receive terminate signal
send final assignment  $P(H)$  to agents

```

Figure 2.1: Center-based Assignment Algorithm

algorithm, one agent c is designated as the *center*. This agent may be a member of A ; it implements the algorithm provided in Figure 2.1. First, it initializes a history variable that keeps track of agents’ responses. It then loops through a series of cycles; each cycle involves constructing an announcement, communicating that announcement, and receiving a bid from the agents.

Definition. [Announcement] An announcement is a pair $\langle P, Q \rangle$ where P is an assignment of the tasks in T' where $T' \subseteq T$ and $Q = (T \setminus T')$ is the set of tasks not contained in that assignment.

The announcement contains both an assignment of items and a set of items. Based on this announcement message, each agent generates a message, which we call a bid that is specified as follows.

Definition. [Bid] After agent a_j receives an announcement message $a(H) = \langle P, Q \rangle$, a bid $B = \{B_1, B_2, \dots, B_l\}$ is a set of pairs where $B_k = \langle T', v \rangle$. The first element of each pair is a set of items $T' = P_j \cup T''$ where $T'' \subseteq Q$ and the second element is the agent’s value for that set of items $v = v(a_j, T')$.

At each iteration of the CBA algorithm’s loop, the center uses the bid history to determine the next announcement. The loop may terminate as a result of a decision made by the center based on the history or because of an external terminate signal. When the loop terminates, the center sends an announcement of the chosen assignment to the agents.

Mediation and combinatorial auctions differ in the algorithm that the center uses to determine the number of iterations allowed, the announcement made at each iteration, and the bids that agents may make. The differences are summarized in Figure 2.2. In a combinatorial auction, the auctioneer’s announcement contains

CBA	Announcement	Bids	Iterations
Combinatorial Auction	$T' = \emptyset$	$ B \leq 2^m$	usually one
Mediation	$T' = T$	$ B = 1$	many
Hybrid	$T' \subset T$	$ B \leq 2^{ T \setminus T' }$	some

Figure 2.2: Overview of CBA algorithms

no assignments, thus $P = \emptyset$ and $Q = T$. Agents may bid on zero or more subsets of tasks in T . In a one-shot combinatorial auction, only one iteration of the algorithm takes place before winners are announced. In Mediation, the mediator announces one possible assignment P . The agents bid on the value of only that one assignment, and the procedure repeats. When time runs out, the mediator announces the best assignment found so far.

Combinatorial auctions give more flexibility to agents as they bid. For example, agents may bid on a certain number of subsets of items that they value the most, or they may bid on each subset of items. There is an exponentially large space (size 2^m) of subsets on which an agent may base its bid. After the auctioneer receives bids, it runs a winner determination [[Sandholm 1999a]] algorithm to find the best assignment based on the bids it has received. In Mediation, the flexibility lies with the mediator and not the agents. The Mediator determines the order in which assignments are announced. Agents have a simple rule for bidding: report the value for the assignment encoded in the announcement that has just been received.

In mediation the flexibility lies with the center; compared to one-shot combinatorial auctions, we claim that Mediation is better suited to real time environments in which there is a limited amount of time to find an assignment. In combinatorial auctions, agents can inform the center which subsets of items are most valuable, thus eliminating time wasted on announcements that are likely to be fruitless. CBA allows for a hybrid approach (see Figure 2.2), in which some items may be assigned while others are open for agents to bid on as in a combinatorial auction. This may allow for agents to achieve the benefits of both methods where the center can fix assignments to a subset of items, and allow agents the flexibility to bid on the rest of the items.

Definition 2.2.2 (Assumptions) *In this chapter we assume (unless otherwise stated) that:*

1. *No inter-agent interactions: that is, we restrict task interactions to those occurring between a single agent's tasks.*

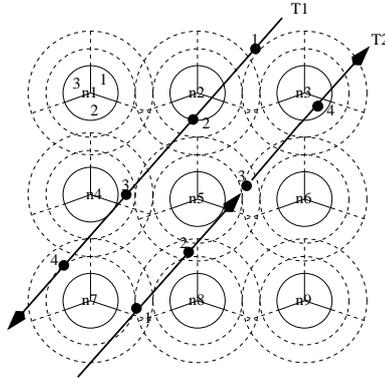


Figure 2.3: Multisensor tracking.

2. *Synchronous and guaranteed communication.*
3. *No faults to the mediator. We relax this assumption later.*
4. *Only pairwise task interactions.*
5. *Task type information is common knowledge to all agents.*
6. *Agents have complete and correct information to make both planning decisions and compute costs.*
7. *The cost functions allow for inter-personal comparisons and can be summed to determine social welfare.*

The distributed sensor challenge problem revisited Figure 2.3 depicts an array of nine doppler sensors. Each sensor has three sectors associated with it, labeled $\{1, 2, 3\}$. A sensor can turn on a sector and take both frequency and amplitude measurements to determine velocity and distance. A sensor can only have one sector on at a time, however. The farther away the target is from the sensor, the lower the quality of the measurement. At least two sensors are necessary for estimating the location of an object; three or more sensors are desirable for obtaining a good-quality estimate. Tasks can interact: for example, sectors require a 2 second warm-up time; therefore, an agent can benefit from tracking two targets in sequence because of the saved warm up time. Finally, two objects appearing in the same sector and at the same time cannot be discriminated.

Tasks can appear dynamically; the figure shows projected paths — based on initial localization, direction and velocity measurements — for two targets, $T1$ and $T2$. The problem is to allocate, in a distributed manner, a set of sensors along the paths of both targets. Each path is discretized into a set of space-time points along the path (indicated in the figure by small dark circles).

In the case of the challenge problem, the varieties of center-based algorithms that we have discussed each have particular advantages and disadvantages. Sequential auctions are attractive because they specify simple bidding rules for agents. Agents (sensors) bid on their expected contribution to a tracking task, which amounts to communicating the agent’s current distance and angle from the task. This suggests that auctions can be used as an allocation mechanism for sensors or resources that *need not remain stationary*. However, one disadvantage of sequential auctions is that they provide no context — in the form of a list of other tasks to which an agent will be assigned in later auctions — on which to base a bid. The context can reflect important interactions that can take place between tasks. With sequential or parallel auctions, agents are compelled to make assumptions about the outcomes of other, related auctions when bidding – assumptions that may turn out to be incorrect. For example, if two tasks that are in sequence are assigned to the same sensor, and if the sensor has that information, it knows it can take advantage of the fact that the sensor’s sector need not be warmed up in preparation for the second task and bid accordingly.

Combinatorial auctions have the advantage of allowing an agent to pick certain bundles of tasks which might interact in a favorable way, as in the example of the previous paragraph. However, as already mentioned, they introduce a bid generation problem. In addition, neither can handle task interactions that might arise at the *group* level. For example, if two agents, a_1 and a_2 are bidding on t_1 and t_2 , respectively, as part of a group task, t , then a_1 might bid on a_1 differently if it knew that a_2 was planning to bid on t_2 . Consider a cooking scenario in which one person is tasked with preparing a particular dish, part of which is to be prepared by another agent. Knowledge of that other agent’s individual abilities (for example, that the agent is particularly good at making a good dressing) can influence the agent’s bid.

There are other issues that arise which will be discussed in this chapter having to do with task re-allocation and providing information to a center that is more informative than simply a single value.

2.3 Negotiation in context

The Mediation algorithm is given in Figure 2.4. Its inputs are \mathbf{P} , A , and an update procedure, an example of which is called Allocation Improvement Mediation (AIM) and is presented in Section 2.3.1. The Mediation algorithm supports making group decisions in general settings. In this chapter we explore the properties

```

function MEDIATION returns an outcome
  inputs:  $P, A, \text{UpdateProcedure}$ 
  let  $b \leftarrow \emptyset, b_{\text{val}} \leftarrow \text{VALUE}(\emptyset)$ 
  loop
     $c \leftarrow$  next value generated by UpdateProcedure
    broadcast  $c$  to  $A$ 
    for each  $a_i$  in  $A$ 
      receive  $msg_i$  from  $a_i$ 
     $c_{\text{val}} \leftarrow \text{VALUE}(msg_1, msg_2, \dots, msg_n)$ 
    if ( $c_{\text{val}} \succ b_{\text{val}}$ ) then
       $b \leftarrow c, b_{\text{val}} \leftarrow c_{\text{val}}$ 
  until (stop signal)
return  $b$ 

```

Figure 2.4: Mediation Algorithm

of Mediation as it applies to the more specific problem of task assignment.

In Mediation, an agent is selected to act as mediator and implements a hill-climbing search in the proposal space, while communicating with the distributed group of agents through a communication channel. Use of the channels is costly in terms of time and perhaps other resources, but it is assumed to be lossless.

The Mediation algorithm proceeds as follows. The mediator initializes a variable b (which represents the best proposal found so far) with \emptyset , along with an initial value denoted $\text{VALUE}(\emptyset)$. Then, it calls an update procedure to generate another proposal c (called the current proposal). That proposal is broadcast to the group. Each agent then responds with a message that is based on the proposal that was broadcast; msg_i denotes the message sent by agent i . The messages are combined to form a value, denoted $\text{VALUE}(msg_1, msg_2, \dots, msg_n)$ ². If that value is preferred to the value for the current b (based on the preference relation \succ), b is updated with the current proposal c .

The algorithm is anytime: it can be halted at any time and will return the best proposal found so far. The proposal stored in variable b is returned as the outcome when the procedure is terminated. Therefore, Mediation is applicable even if agents do not know in advance how much time they will have to negotiate.

² VALUE may return a real number (i.e., when objective is social welfare) or a vector (i.e., when objective is Pareto efficiency).

With a choice of agent messages and VALUE function that satisfies the properties in Theorem 2.1, Mediation implements a hill-climbing search in the space of proposals, with objective function f .

Theorem 2.1 *Let msg^p denote the messages returned by agents in G after the mediator broadcasts proposal p . Assume $\text{VALUE}(\text{msg}^p) \succ \text{VALUE}(\text{msg}^q)$ if and only if $f(p, G) > f(q, G)$, and $\text{VALUE}(\text{msg}^p) \succ \text{VALUE}(\emptyset)$ if and only if $f(p, G) > f(\emptyset, G)$. Then after each iteration of the loop in the Mediation Algorithm, b contains the best proposal (according to the objective function) generated so far by the update procedure that is at least as good as \emptyset .*

Proof. Assume that the update procedure generates a proposal c_0 such that $f(c_0, G) > f(b, G)$ (i.e., it is the best proposal generated so far). Therefore, $\text{VALUE}(\text{msg}^{c_0}) \succ \text{VALUE}(\text{msg}^b)$ by assumption, and b is updated with c_0 . $f(b, G)$ will never be lower than $f(\emptyset, G)$ because for all c_1 generated such that $f(\emptyset, G) > f(c_1, G)$, it is the case that $\neg(\text{VALUE}(\text{msg}^{c_0}) \succ \text{VALUE}(\emptyset))$, by assumption, and no update occurs.

The example described in Section 2.3.2 implements agent messages and VALUE function that is consistent with the assumptions of Theorem 2.1. It follows from the theorem that if \mathbf{P} is finite and the update procedure eventually returns every proposal in \mathbf{P} , given a sufficient number of iterations of the Mediation algorithm, the final value for b will be a proposal that maximizes the objective function f .

Mediation does not impose a particular search order on the problem but rather supports update rules that can be designed to search the space of proposals in a variety of ways. A simple, uninformed update procedure returns a proposal randomly selected with replacement from the set of all proposals. The mediation algorithm with such an update procedure is called *Random Mediation*. The next section provides an example of another update rule that can be used with Mediation.

2.3.1 Allocation Improvement

Allocation Improvement defines an update procedure for Mediation that supports task allocation domains. The procedure is sketched in Figure 2.5. The first proposal p is chosen randomly from \mathbf{P} ; it provides a context, from which subsequent proposals are generated. For example, it might return $\langle \{t_2\}, \{t_0, t_1\} \rangle$ which corresponds to the proposal where agent 0 is assigned to task 2, and agent 1 is assigned to tasks 0 and 1. The advantage of this context is that it is common to all agents and it ensures that each task is assigned to an agent.

```

Let  $p \leftarrow$  a random element of  $\mathbf{P} - \{\emptyset\}$ ; return  $p$ 
for  $i = 1 \dots |T|$ 
  for  $t \leftarrow$  every set of tasks of size  $i$ 
    for  $a \leftarrow$  every possible assignment of agents in  $A$  to tasks in  $t$ 
       $q \leftarrow$  substitute  $a$  in  $p$ ; return  $q$ 
      if  $q_{\text{val}} \succ p_{\text{val}}$  in Mediation, then  $p \leftarrow q$ .

```

Figure 2.5: Sketch of the Allocation Improvement Update Procedure

In subsequent iterations, the procedure returns proposals that result from making substitutions in p for i -tuples of tasks where i goes from 1 to $|T|$. Substitutions for each i -tuple of tasks is made sequentially with each permutation of agents in lexicographic order, while maintaining the allocations for the other tasks. p is always maintained to correspond with the best proposal seen so far (b from the mediation algorithm).

In the example, the next proposal chosen will involve substituting agent 0 as the agent to perform task 0 (e.g., $\langle \{t_0, t_2\}, \{t_1\} \rangle$ will be returned). Then, agent 1 will be substituted as the agent to perform task 0, and so on. This procedure is then repeated for each pair of tasks in lexicographic order (i.e., $(t_0, t_1), (t_0, t_2), (t_1, t_2)$). Finally, every possible substitution (i.e., every element in $\mathbf{P} - \{\emptyset\}$) is sequentially returned.

Overhead The Allocation Improvement procedure runs in constant space, is guaranteed to eventually return every element of \mathbf{P} (in the last stage), but involves an overhead cost in running time because it may return one proposal more than once. In a task allocation problem, the size of \mathbf{P} is $|G|^{|T|}$. The Allocation Improvement procedure returns a total of $(1 + |G|)^{|T|}$ proposals. For example, with 20 agents and 10 tasks, the Allocation Improvement algorithm has an overhead of 62.9% because it returns 62.9% more proposals than there are elements of \mathbf{P} before it terminates. By increasing memory bounds, the overhead can be eliminated because each proposal can be checked for redundancy before it is returned.

2.3.2 Experimental Evaluation

We performed a preliminary evaluation of mediation in a simplified version of the Challenge Problem in which we map groups of three sensors into a single entity, so that a proposal is of the standard form discussed earlier. An equivalent alternative is to allow for more complex proposals. For example, $\langle \{a_1, a_3, a_4\}, \{a_2, a_5\}, \dots \rangle$

in which sets of agents are allocated to a single task. Agents represent n such hidden sensors of varying quality that are randomly distributed within a 100-meter by 100-meter square. An object appears at a point at the top edge of the square and makes a 10-second trip at constant velocity through the square, exiting at the bottom edge. One sensor is required to track the object every two seconds during its journey.

Each agent’s value function is derived from encoding the knowledge that benefits to the group are higher when each measurement is taken from as close as possible to the object with the highest quality sensor. The agents also incur costs when taking the measurement because there is assumed to be a 2-second warm-up time, but once warmed up, the sensor can take a measurement every two seconds. The cost function exhibits task interaction as a result of this warm up time: costs will be lower if the same sensor takes consecutive measurements. Benefits and costs are quantified by a local utility function known to each agent. Agents do not know the utility or even the location of other agents.

The experiments were designed to test the hypothesis that the Allocation Improvement procedure is an effective strategy for task allocation in the tracking domain. AIM is expected to perform well in the early stages of Mediation in this domain because it searches for gains that come from assigning the best agent to each task (which requires a relatively small number of iterations) before it tries to find improvements based on assignments for pairs of tasks, triples of tasks, and so on.

The Mediation algorithm was implemented with agent message and value functions that adhere to the assumptions presented in Proposition 2.1. In msg_i , agent G_i reports the value associated with its quality and expected distance from the object, less the warm-up costs, for the set of tasks to which it has been allocated in the latest proposal. $VALUE(msg^p)$ is chosen to be the sum of the values reported in each agent’s message.

AIM was compared to other instances of the Mediation algorithm using two different types of update rules. *Full Search* simply returns successive elements of \mathbf{P} as they would be explored in a depth-first search. *Random Mediation* returns a random element of \mathbf{P} at each iteration. In Section 2.5 we discuss ways to make Mediation’s search smarter.

Results: Each negotiation method was run on 100 problem instances. The first experiment was run with a group of 4 agents, which implies $|\mathbf{P}| = 1025$ (including \emptyset). Figure 2.6 shows the social welfare of proposal b after each iteration of the

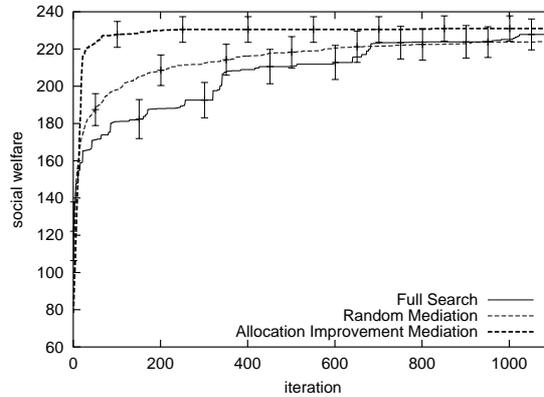


Figure 2.6: Comparison of Mediation algorithms in the 4-agent sensor domain

Mediation Algorithm, using the three different update rules, along with 95% confidence intervals. The maximum average social welfare attainable differed slightly for the three algorithms since each method was run on 100 different sets of problem data. The results strengthen the hypothesis that Allocation Improvement is an effective strategy in this domain, especially when agents may search through only a small number of proposals before they require an outcome and must act.

After just 30 iterations, the average social welfare of the group is 219.3 using AIM versus 181.0 using Random Mediation. The highest average social welfare attainable by exhaustive search is approximately 228. AIM allows agents to quickly capture the gains in increased social welfare that results from assigning each task to the closest agent by delaying the search for gains based on task interaction. As expected, after a large number of iterations, any update procedure will have exhausted the search space, and all procedures will have found outcomes of similarly high quality.

The results shown in Figure 2.7 is for groups of 20 agents. With 20 agents, \mathbf{P} is of size 3.2 million, and a full search of the space is prohibitively costly for agents that must act quickly. The graph shows the average social welfare after the first 1000 iterations of each Mediation algorithm across the 100 problem instances. The focus is on the first 1000 problem instances because the negotiation must be short due to the real time nature of the problem. Random Mediation and full search perform fairly well in the very early stages of the algorithm. However, after about 65 iterations or 0.002% of the search space, AIM performs significantly better than either of the other two methods ($p < 0.01$). The superior performance of AIM is more pronounced in the figure that that shows the results of this larger

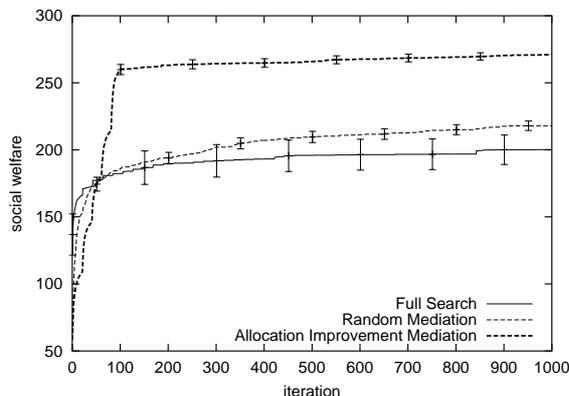


Figure 2.7: Comparison of mediation algorithms in the 20-agent sensor domain

problem because only a very small percentage of the search space is shown. The gains of AIM come very early in the negotiation, as expected.

The Random and Full Search update rules outperform Allocation Improvement at the very early stages of the search. This effect may be due to the sequential update procedure of Allocation Improvement (i.e., all agents are considered for a given task before the allocations to the other tasks are searched).

2.4 Combinatorial task allocation

To study the impact of general task interaction on the task allocation problem we consider a set of synthetic multi-agent domains. The simplification of the challenge problem presents a specific type of task interaction, namely subadditive cost functions that result from savings in warm-up time. A particular domain is associated with a certain *interaction probability*, denoted by ip . This probability quantifies the extent to which the tasks interact in an agent's cost function.

A domain in which $ip=0$ describes a situation in which there is no task interaction. A domain in which $ip=1$ describes a scenario where tasks have arbitrary interaction.

The following algorithm is used to determine each agent's cost function. First, construct an undirected graph with m vertices; each vertex corresponds to a task. Second, assign a set of edges to the graph. Each pair of vertices is connected by an edge with probability ip . A set of tasks interacts if and only if the subgraph containing only the vertices that correspond to the tasks is connected.

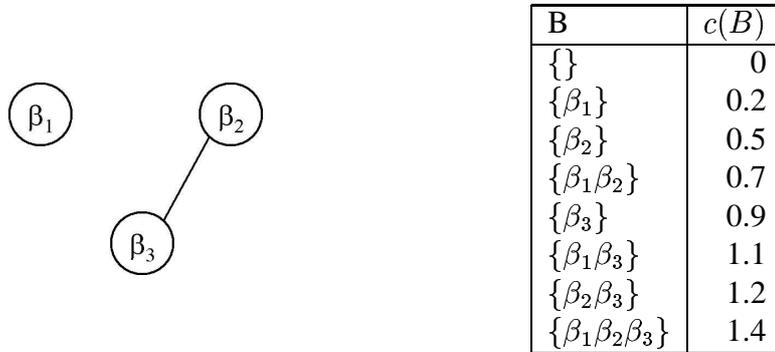


Figure 2.8: Task interaction graph and cost function for Agent 1

For the purposes of experimentation, the cost to an agent of performing a set of interacting tasks B_I is assigned randomly, according to a uniform distribution on real values from 0 to $|B_I|$. The cost of performing a set of tasks in which some of the tasks do not interact is the sum of the cost of performing each interacting set of tasks.

Figure 2.8 provides an example task interaction graph and cost function, where β_2 and β_3 interact. As a result, execution of both tasks has a different cost than the sum of costs of the individual tasks (i.e., $1.2 \neq 0.5 + 0.9$). In this case, β_2 and β_3 exhibit a positive interaction because the cost of performing both is lower than the sum of the costs of performing each task alone.

When a given task is being awarded under sequential or parallel auctions, outcomes of other task auctions may be unknown. When tasks interact, this can result in task allocations that are inefficient. That is, there may be another task allocation that has lower cost.

Ideally, a combinatorial auction would solve this inefficiency by allowing all task assignments to be made as a package. Combinatorial auctions for task allocation pose two new problems for a group of agents to consider. First, an agent must decide which bundles to bid on, called the *bid generation problem*. Second, based on those bids, the auctioneer must determine which set of task allocations corresponds to the minimum total cost for the group of agents.

The second problem has been studied extensively in the AI literature [Sandholm 1999a, inter alia]. Although winner determination is NP-complete, Sandholm and Suri [Sandholm and Suri 2000] have developed an anytime search algorithm that uses the set of bids to focus the search on fruitful areas of the search space and can provide approximate solutions. They also point out that typical combinatorial auctions do not allow agents to express bids for tasks with negative interaction

(subadditive preferences). The winner determination algorithm is significantly complicated by the inclusion of methods for allowing such bids.

Bid Generation: The problem of bid generation has not received much attention, and is not trivial. We define a *relevant bid* as one that cannot be inferred as an additive combination of an agent’s other bids. For example, if an agent bids a cost of 0.3 units for $\{\beta_1\}$ and 0.4 units for $\{\beta_2\}$, an addition would imply a bid of 0.7 units for tasks $\{\beta_1, \beta_2\}$. A bid of say, 0.5 units for tasks $\{\beta_1, \beta_2\}$ would be a relevant bid. To achieve a minimum cost allocation, it is important to generate all relevant bids (Theorem 2.2). However, it may be infeasible for an agent to generate all of its relevant bids (Theorem 2.3). Thus, implementing a combinatorial auction for task allocation that presupposes bid generation may be infeasible.

Theorem 2.2 *If all relevant bids are not generated, then the optimal task allocation solution computed by combinatorial auction winner determination may be suboptimal.*

Proof. (by counterexample). Assume a 2-agent group action that decomposes into two tasks and the cost functions are given by: $c_1(\{\beta_1\}) = 0.5$, $c_1(\{\beta_2\}) = 1.0$, $c_1(\{\beta_1, \beta_2\}) = 0.6$ and $c_2(\{\beta_1\}) = 1.0$, $c_2(\{\beta_2\}) = 0.5$, $c_2(\{\beta_1, \beta_2\}) = 1.6$. Assume the relevant bid by Agent 1 of 0.6 units for $\{\beta_1, \beta_2\}$ is not generated. The outcome of a combinatorial auction would be the allocation of β_1 to Agent 1 and β_2 to Agent 2 for a total cost of 1.0. However, the optimal allocation would be to assign both tasks to Agent 1, for a total cost of 0.6.

Theorem 2.3 *If $ip=1$, the number of relevant bids for each agent is $2^m - 1$.*

Proof. The cost of a task set that is not fully connected can be determined by the sum of its connected parts. Therefore, a bid is relevant if and only if its tasks are fully connected. The task interaction graph is fully connected ($ip=1$). Therefore, the number of fully connected sets is $\mathcal{P}(B_T)$, which has size 2^m , less the bid on the empty set which is defined as 0.

2.4.1 Incremental Task Allocation Improvement Algorithm

In this section, we present an anytime algorithm, called *Incremental Task Allocation Improvement* (ITAI), that does not require a bid generation phase as input. Agents incrementally reveal their costs for bundles of tasks.

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

Figure 2.9: Anytime Algorithm for Task Allocation

The algorithm is summarized in Figure 2.9. One way of performing the initial allocation step of Step 1 quickly is by sequential auction. As discussed above, task interaction may cause this allocation to be suboptimal.

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

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

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

Theorem 2.4 *The algorithm is guaranteed to find the optimal task allocation.*

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

Time Complexity: Similar to the general iterative deepening search algorithm, ITAI incrementally expands the scope of its search for the optimal task allocation. The time spent on task allocation is the sum of the time spent generating the initial allocation, plus the time spent improving the allocation. As Theorem 2.5 illustrates, the complexity of the algorithm is the same as an algorithm that performed optimal allocation of all tasks in a single step.

Theorem 2.5 *Assuming an iteration of the improvement phase that allocates i tasks takes $O(n \cdot 2^i)$ time, the running time of the improvement phase is $O(n \cdot 2^m)$.*

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

$$O\left(\sum_{i=2}^m n \cdot 2^i\right) = O(2n(2^m - 2))$$

which is $O(n \cdot 2^m)$.

2.4.2 Empirical Evaluation

The generalized task interaction domain described earlier is used to generate results of ITAI. Figure 2.10 illustrates the movement of the total cost attained by the successive sequence of allocations generated by the algorithm. The experiment was run in a 2-agent, 10-task domain. Therefore, there were $2^{10} = 1024$ total possible task allocations.

The three separate experiments correspond to domains with $ip=0, 0.5, \text{ and } 1$ and results shown are averages over 20 instantiations of each domain. Due to slight variations in assigned cost functions in the three cases, the costs have been normalized, with the cost of the initial allocation corresponding to a value of 1 on the y-axis. The initial allocation was made by sequential auction, with no look-ahead (agents computed their bid for a task assuming they would execute only those tasks already allocated to them). The optimal allocation algorithm used

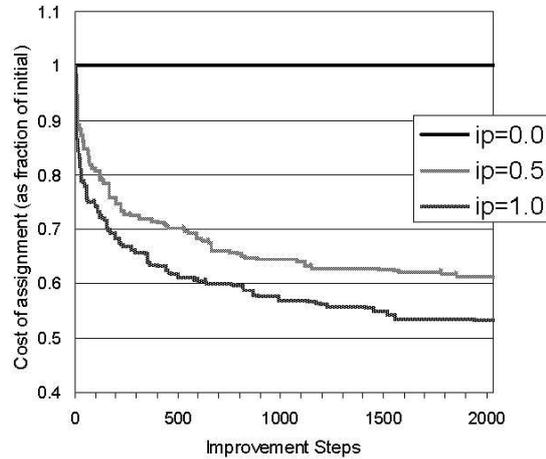


Figure 2.10: Improvement of task allocation cost under ITAI.

in the improvement phase iterated over all possible task allocations, collecting a bid for each from each agent, and updating the allocation with that which has the minimum sum of bids. The generation and evaluation of bids for each hypothetical allocation is referred to as an *improvement step*.

Since the choice of the edge to add in Step 3 is nondeterministic, the number of steps required for the improvement phase of the algorithm to complete varied. The mean was 1765; the maximum was 2035, at which point the allocation was guaranteed to be optimal (by Theorem 2.4).

As expected, for $ip=0$, the improvement phase did not lower the cost of the allocation because all tasks were independent for both agents. When task interaction was introduced, the improvement phase of the algorithm performed well. For $ip=0.5$, after 196 steps, the improvement phase had generated an average of 25% in cost savings, which corresponds to over 64% of the potential total cost savings of 61%. For $ip=1$, after only 93 steps, the improvement phase had generated an average of 25% in cost savings, which corresponds to over 53% of the potential total cost savings of 53%.

2.5 Dynamic negotiation

The development of negotiation protocols to support the distributed allocation of tasks or resources within a multiagent system has assumed that the set of issues over which agents negotiate are defined in advance. Most realistic problem domains, however, are dynamic: while agents are negotiating over the distribution of a set of tasks, for example, new tasks or resources might appear or disappear. Existing protocols either require that the allocation of a new task be postponed until the current negotiation is complete or require that the negotiation be interrupted and re-started in the context of the augmented set of tasks. Adopting the first option neglects the potential for exploiting possible positive interactions between the old set of tasks and the new task, since they are allocated separately. In the second option, all of the work performed in the first negotiation is lost and, in fact, there can be no guarantee that the process will ever converge since new tasks could continue to appear. Ideally, it should be possible to adapt the ongoing negotiated allocation to the new task. Similarly, agents not currently involved in a negotiation might become free to offer their services on the task being negotiated. If an agent becomes disabled, it should also be possible to repair the current allocation as reflected by the current state of the negotiation rather than restart the negotiation from scratch.

The problem of dynamic negotiation is reminiscent of that faced by conventional single-agent planning systems in the 1980's: then it was assumed that complete executable plans could be generated before execution commenced; the introduction of new goals during the planning process required a complete replanning. It was discovered that such a view was unrealistic for any but simple toy domains.

In this section, we examine negotiation in team-based settings where tasks are not necessarily limited to primitive actions but are instead richly described entities [[Tambe 1997, Rauenbusch 2003, Ortiz and Hsu 2002]]. As before, our concern is in domains where full exchange of information among agents is not feasible; hence, mediators will have to operate in the face of incomplete information. In addition, we assume that the agents participating in such teams will be concerned with the means for executing only their individual tasks; hence, the mediator will normally not concern itself over local planning decisions of any one agent. The main idea behind our solution involves the exchange of local information in the form of a description of positive and negative interactions between task *types*. By focusing on task types, mediators can use prior bids in a predictive fashion to prune the search space of possible future proposals.

Examples of domains for which the approach we describe in this paper are

applicable include distributed transportation scheduling and distributed sensor scheduling. For generality we present our results within the first domain: the set of tasks might include delivery and pickup tasks, each with parameters for source, destination, time, object to deliver, truck for delivery, and path to destination. The chosen path and the identity of the truck correspond to planning decisions made at the local agent level. Tasks in the sensor domain map almost directly to tasks in the transportation domain. In the sensor domain, multiple sensors are necessary for estimating the location of an object. As we have already discussed, tasks can interact: for example, if sectors require a warm-up time, an agent can benefit from tracking two targets in sequence because of the saved warm up time. Corresponding to a $delivery(Agent, Destination, Object, Truck, Time)$ task description in the transportation domain might be a task in the sensor domain represented as $track(Sensor, UntilTime, Object, Sector, Time)$.

Each executable task description has one or more associated types taken from the set $Types$. We assume a function $type : T \rightarrow 2^{Types}$, where T is the set of tasks. For example, an executable task description might be of the form $t = deliver(Agent25, Propane, Warehouse)$, where type information is specified by $type(t) = \{delivery_to_Warehouse, delivery_of_Propane\}$. Bidders use such information to communicate task interactions to the mediator (see next section). We add the notion of agent *capacity* to the problem statement: let $c : A \times R \rightarrow \mathcal{N}$, an agent's current capacity, e.g., $c(a_5, t) = 3$ means the agent a_5 can take on 3 more tasks.

In this section, we imagine that each agent's state (from the set S of states) is structured in some way. For example, the agent might store information regarding its current location, past proposals that it has received, a cost/utility function, and its current capacity. The mediator, in turn, has only partial knowledge of any individual agent; the mediator records task preference information of non-mediator agents in what we call an *interaction table*, \mathcal{I} , and capacity status information supplied to it by way of rich bids by a bidder. The task interaction table is built up by the mediator for each agent and records interactions between task types. For example, agent a_i might report the following interactions $\{t_1 + t_4, t_1 || t_3, t_1 - t_5, t_4 \bowtie t_6\}$, where $t_j \in Types$, indicating that types t_1 and t_4 interact positively, t_1 and t_3 are independent (that is, their costs are additive), t_1 and t_5 interact negatively and t_4 and t_6 conflict (cannot be executed together). In a delivery domain, t_4 might correspond to a delivery of flammable goods task type and t_6 to a delivery of explosive materials task type. The stated task information constrains task assignments involving delivery of both types of materials. The alternative involving the comparison of fully specific executable tasks would be less useful.

For example, if the actual task allocations were: delivery of Propane-j35 to Logan at 2pm and delivery of fireworks to Logan at 2:30pm, reporting a negative interaction between the two would have little value in helping to prune the space of future proposals, because of their specificity.

The capacity information might indicate that a particular agent (e.g., truck) can fit one more crate. However, the cost for delivery of that crate might depend on the type of task with which it is associated (e.g., the destination, the crate's weight, or path).

Definition 2.5.1 (A Dynamic Negotiation) Let $t \xrightarrow{i} a$ refer to the set of tasks, t , assigned to agent a in proposal P_i . We augment the definition of a task allocation system, M , to include capacity status: $M = \langle A, T, S, u, c, \mathcal{P} \rangle$. A dynamic mediated negotiation, N , is a sequence of messages: $\langle P_1; B_2; \dots; \Delta_j; \dots; P_k; B_{k+1} \rangle$ such that each $P_i \in \mathcal{P}$ for $i < j$ and each $B_i = \{bid_1, bid_2, \dots, bid_m\}$ corresponds to a set of rich bids from each agent in P_i . Each pair of indices for P and B correspond to a round in the mediation. Each Δ_j represents a pair, $\Delta_j = \langle T_{new}, A_{new} \rangle$ of new tasks and agents; more than one Δ can appear in any sequence. The social welfare, W , for some proposal, P_i , is defined as:

$$W(P_i) = \sum_{a \in P_i} u(a_i, t \xrightarrow{i} a)$$

2.5.1 Rich bids

Rather than reducing a bid for some task to a single value, Dynamic Mediation makes use of a richer bid format which allows a bidder to compactly exchange relevant information about its local state to the mediator. The mediator can then use that information during its search. We refer to the set of resources (agents) and tasks collectively as the *negotiation space*. The negotiation space might change because of either a *negotiation event* (the mediator considers a new resource) or a *domain event* (a new task appears).

The format of a bid for a_i at time t for proposal P is:

$$\left\langle \sum_{e \in p \xrightarrow{i} a} u(a, e), \mathcal{I}, c(a, t) \right\rangle$$

where the first argument represents the agent's bid for each task in the proposal, \mathcal{I} represents a list of relevant task interactions from a_i 's point of view for a subset

of the tasks covered in P , and $c(a_i, t)$ reflects agent a_i 's current capacity. This is by no means the only alternative possible. One could also consider providing counterfactual values for each task that is reported to interact in a negative way with another task. Another alternative is to communicate a qualitative preference to some other task [[Doyle *et al* 1991]]. Such information, however, does not capture the *reason* for the preference: this is captured compactly in task interaction statements. Since we were most interested in minimizing the amount of information shared with the mediator, we chose the format shown above.

Task interaction semantics and bid generation

A positive interaction between tasks, t_1 and t_2 reflects task complementarity in terms of an underlying cost function, u , with the a superadditive property for some agent a , $u(a, \{t_1, t_2\}) < u(a, \{t_1\}) + u(a, \{t_2\})$, for example, common delivery destinations. A negative interaction corresponds to task substitutability (subadditivity), $u(a, \{t_1, t_2\}) > u(a, \{t_1\}) + u(a, \{t_2\})$, for example, a delivery requiring two separate trips.

The framework we have described introduces the following *bid generation problem*: how should the agent decide on which potential interactions to communicate to the mediator? The difficulty is that a statement by agent a_i of $t_1 + t_2$ might hold under context, C (i.e., under some collection of other task assignments for a_i under the current proposal), while under another context, C' , the interaction $t_1 - t_2$ might instead hold. We are currently exploring a general approach to handle this problem by making use of a task abstraction hierarchy and the observation that a task t , in context, C , can be captured as a separate task type itself [[Ortiz 1999]]. The abstraction hierarchy represents individual tasks and tasks performed while performing other tasks. The bidder picks the most abstract task descriptions which stand in the indicated positive or negative interaction. In this way, the task interaction table is always consistent. In our actual algorithm, we use less detailed type information and use task interactions as heuristics to increase the probability of finding a solution.

Dynamic mediation algorithm

Figure 2.11 presents the dynamic mediation algorithm. Recall that the algorithm implements an iterative hill climbing search through the proposal space, keeping track of the *best proposal*, b , found so far. At each step the mediator selects and communicates the *current proposal*, c , to the agents in the group. This section ex-

```

function DYNAMICMEDIATION returns an outcome
inputs a set of tasks  $T$ , set of agents  $A$ 
let  $b \leftarrow \emptyset$ ,  $b_{\text{val}} \leftarrow \text{VALUE}(\emptyset)$ 
let Interaction Table  $it \leftarrow \emptyset$ 
loop
   $c \leftarrow \text{GETNEXTPROPOSAL}(T, A, it)$ 
  broadcast  $c$  to  $A$ 
  for each  $a_i$  in  $A$ 
    receive  $\text{bid}_i$  from  $a_i$ 
    store interactions in  $\text{bid}_i$  in  $it$ 
   $c_{\text{val}} \leftarrow \text{VALUE}(msg_1, msg_2, \dots, msg_n)$ 
  if ( $c_{\text{val}} \succ b_{\text{val}}$ ) then
     $b \leftarrow c$ ,  $b_{\text{val}} \leftarrow c_{\text{val}}$ 
until (stop signal)
return  $b$ 

```

Figure 2.11: Dynamic Mediation Algorithm

tends the algorithm to support dynamics by adding maintenance of an interaction table to leverage information received from bidders and to focus the search.

Each agent then responds with a bid that is based on the proposal that was broadcast; bid_i denotes the bid sent by agent i . The information about interaction among task types that is provided in the bid is stored in the *Interaction Table*. The mediator uses the information contained in the interaction table to focus its search. In each round of mediation, the interaction table is used to construct a probability distribution over the set of agents for each task in the proposal. The mediator focuses the search by using the interaction information previously provided to it

```

function GETNEXTPROPOSAL returns a proposal
inputs set of tasks  $T$ , set of agents  $A$ , Interaction Table  $it$ 
let  $P \leftarrow \emptyset$ 
for each task  $t$  in  $T$ 
   $pd \leftarrow \text{GETPROBABILITYDISTRIBUTION}(t, P, A, it)$ 
   $a_t \leftarrow$  agent chosen randomly according to  $pd$ 
   $P \leftarrow P \cup \{\text{assign}(t, a_t)\}$ 
return  $P$ 

```

Figure 2.12: Update procedure for Dynamic Mediation.

function GETPROBABILITYDISTRIBUTION
inputs task t , proposal P , set of agents A , InteractionTable it
returns a probability distribution over A
for each agent a in A
 for each task u in $\{u \mid assign(u, a) \in P\}$
 $I \leftarrow I \cup getInteraction(t, u, it)$
 if $\exists i \in I, i \in \{\times\}$ then $score_a \leftarrow 0.0$
 else if $\forall i \in I, i \in \{+, ?\}$ then $score_a \leftarrow 2.0$
 else if $\forall i \in I, i \in \{-, ?\}$ then $score_a \leftarrow 0.5$
 else $score_a \leftarrow 1.0$
for each agent a in A
 $pd(a) \leftarrow score_a / \sum_{b \in A} score_b$
return pd

by some agent to adjust the likelihood that the agent will be assigned a given task. For example, if the mediator expects a positive interaction between two tasks for some agent, those two tasks should be more likely to be assigned to that agent. If the mediator knows that two tasks conflict for a given agent, that agent should not be assigned to both tasks. An example of an algorithm for the generation of probability distributions is given in Figure 2.12. This algorithm implements a stochastic, heuristic-based search that weights task assignments according to the information contained in the variable, it .

The mediation algorithm supports dynamic adjustments to the set of tasks to be negotiated. Since the interaction table stores information about task types, that information may be used to determine probability information for new tasks that arrive after a negotiation begins. The mediation algorithm has the following anytime property which makes it applicable even if agents do not know in advance how much time they will have to negotiate.

Note that without the capacity parameter, as the system tries to respond to new tasks, it can eventually become saturated and thrash. By using capacity information the mediator can signal that execution should begin while certain new tasks are postponed. This can be accomplished by simply adding a line to the *getProbabilityDistribution* function that assigns 0 probability for agents over capacity.

We assume that the mediator stores all of the interaction information from each bidder in a separate table. The memory required for this table grows linearly in the number of interactions reported. In the algorithm, the search space is either expanded by adding a new task or a new resource/agent or narrowed (deleting a

```

function GETINTERACTION
inputs task  $t$ , task  $u$ , InteractionTable  $it$ 
returns one of  $\{+, -, \times, ?\}$ 
interaction  $\leftarrow ?$ 
for each  $t_t \in \text{type}(t)$ 
  for each  $u_t \in \text{type}(u)$ 
    if  $((t, u, \times) \in it)$  then return  $\times$ 
    if (interaction= $\text{none}$ )
      if  $((t_t, u_t, +) \in it)$  then interaction $\leftarrow +$ 
      if  $((t_t, u_t, -) \in it)$  then interaction $\leftarrow -$ 
    if (interaction= $+$ )
      if  $((t_t, u_t, -) \in it)$  then return  $?$ 
    if (interaction= $-$ )
      if  $((t_t, u_t, +) \in it)$  then return  $?$ 
return interaction

```

Figure 2.13: Example Construction of Probability Distribution

task or resource). The mediator can combine steps (for example, removing a task while adding a resource).

Task contention, team composition and fault tolerance

The notion of task contention in the challenge problem is essentially that of a task conflict, as described above. Consider Figure 2.3. We will use the notation S_n/s to refer to sector s of sensor n and the notation t_p^r to refer to the point p on projected path r and t^r to refer collectively to the tasks on path r . Suppose a negotiation involving task t^1 has been ongoing for n rounds with the current best proposal, $P1 = (S2/1, S2/2, S5/3, S7/3)$. Target t^2 then appears and is projected to follow the path shown. Suppose that the best allocation for t^2 , assuming that there were no conflicts, would be $P2 = (S7/1, S8/3, S5/1, S3/2)$. However, if tasks t_3^1 and t_3^2 occur at the same time, a conflict involving sensor $S5$ will result. We refer to this as an instance of *task contention*. There are two cases to deal with. In the first case, if we assume that the same mediator is coordinating the negotiation for both targets, then the mediator will be aware of the commitment of $S5$ to task t_3^1 . If it is aware of the contention between tasks t_3^1 and t_3^2 , then it can adjust its proposal to the group accordingly to avoid a conflict. However, if some other agent is acting as mediator for t^2 , or is not aware of the task contention, then that

mediator might very well propose $P2$, the allocation resulting in task contention. In very large domains, with hundreds or even thousands of nodes, it is infeasible to have a single agent that will centrally coordinate all negotiation. Therefore, agent $S5$ might respond to $P2$ with the bid, $\langle -\infty, t_3^1 - t_3^2 \rangle$ indicating the conflict and its source. To resolve this conflict, the mediator may add an option (i.e., enlarge the group of agents) and propose $(S7/1, S8/3, S6/3, S3/2)$.³ The key point is that the contention cannot be resolved on the basis of local information alone (such as a bid involving a single value); hence the richer bid format we have proposed.

Problems having to do with forming teams with the proper mix of sensors are also dealt with at the mediator level, using rich bids. For example, if 3 agents are associated with a task in a proposal and one of the agents, a_i , bids a low value (such as a short duration measurement) because it is committed to serving on another tracking team consisting of 5 agents, during an overlapping interval, then agent a_i can relay that information to the mediator (this assumes there are multiple mediators) who can then make the determination to perhaps have the agent drop the original commitment, since the new task is of higher priority (the marginal gain from the agent's contribution on the committed task is lower than that on the new task involving a smaller team).

In response to a system fault, such as the loss of an agent (some Δ), the mediator will look for substitute agents. The Mediation algorithm has also been made tolerant to faults to the mediator agent; this extension is not shown in the figure, however. The solution is simple: instead of communicating only with the mediator, agents broadcast their bids to the entire group. If the mediator is disabled, a new one can be chosen; it will have a current record of the negotiation and can proceed where the disabled mediator left off.

2.5.2 Experimental results and evaluation

The value of rich bids and the interaction table as well as the effectiveness of mediation in dynamic environments were evaluated with simulations. We were primarily concerned with studying the effectiveness and feasibility of communicating rich bids, and the incorporation of task interaction information into the interaction table. The domain that we developed had instances of the three types

³In the actual multisensor domain we have described, there are also instances in which the contention can only be resolved if a node switches from one sector to another, halfway between two points on the projected target path, thereby lowering the quality of the measurement for both targets. This complication can be dealt with by partitioning the projected track in a more fine grained way.

of interactions that we defined in our model: positive, negative, and conflict. It is a simplified delivery problem in which agents are assigned tasks of delivering goods to their destinations. Each task was defined by two attributes: type of good and destination. Goods can be one of three types: propane, electrical, or food. There were two possible destinations for each task.

The cost incurred by an agent in executing a set of tasks was determined in advance and remained constant throughout the experiments. First, the cost of delivering one package to a single destination was chosen randomly from a uniform distribution from 0 to 20. The cost of performing each subsequent task was chosen from a uniform distribution from 0 to the cost of performing the previous task. Thus, as more tasks were given to an agent, the additional cost of delivery decreased monotonically. Second, the cost to an agent that is responsible for delivering packages to both locations is double its cost for delivering to one location. Third, no agent can be responsible for delivering both propane and electrical goods.

Agents' rich bids report the interaction among tasks that an agent is assigned. In our experiments, an agent reports all interaction information for each pair of tasks in the set of all tasks to which it is assigned at each iteration. Conflict interaction is reported for each pair of propane and electrical tasks. Positive interaction is reported for each pair of tasks delivered to the same location. Negative interaction is reported for each pair of tasks delivered to different locations.

Figure 2.13 illustrates the probability distribution generation procedure used for the experiments. Tasks that are known to conflict are never given to the same agent. Tasks that have positive interaction are more likely to be given to the same agent; tasks that have negative interaction are less likely.

The *getProbabilityDistribution* function creates a probability distribution over the agents to assign to a given task. The probability of assigning an agent to any given task is affected by the tasks to which it is already assigned and by the information stored by the mediator in the interaction table. The function uses a system for scoring agents' likelihood of assignment to the task. If the task to be assigned is known to conflict with a task type that is already assigned to an agent, assignment of that agent to the task is given a score of zero. Agents assigned to tasks with types that interact positively with the current task are given a score of 2. Agents assigned to tasks with types that interact negatively are given a score of 0.5. Other agents are given a score of 1. The scores are normalized to create a probability distribution over the agents.

Since each task can be associated with more than one task type (e.g., delivery location and type of good), there may be inconsistent interaction information

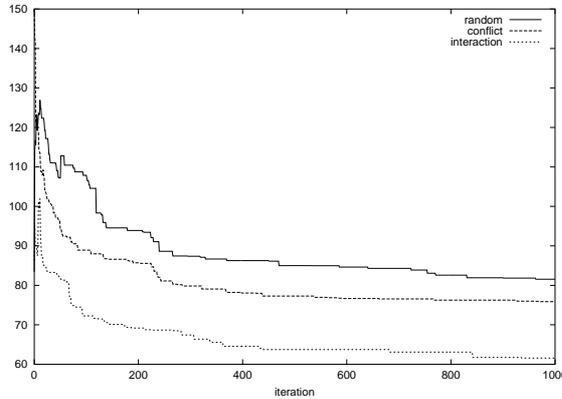


Figure 2.14: Results for static mediation.

stored in the interaction table for interaction between tasks. In this experiment, when inconsistent information is present for a pair of tasks, the function proceeds as though no useful interaction information exists in the *getInteraction* function. Another source of inconsistency arises because an agent may have been assigned to several tasks, which interact in different ways with the current task. Again, the mediator in these experiments assumes that no useful interaction information exists.

The goal of the first simulation was to evaluate the effectiveness of using the task interaction table for making task assignments. The experimental data was generated with the delivery domain described above using four agents and ten delivery tasks. Thus, the search space of all possible proposals was of size 4^{10} , or approximately 10^6 .

Figure 2.14 illustrates the average lowest cost attained in 15 trials by each of the first 1000 rounds of mediation using three separate update procedures: random, conflict and interaction. *Random* ignores the interaction table and generates a random sequence of proposals. *Conflict* uses the interaction table to eliminate assignment of tasks to agents where known conflicts exist. *Interaction* is the update procedure described above that makes a probability distribution based on reported positive, negative and conflict interaction.

The results verify our hypothesis that interaction information and rich bids have significant benefit. For instance, after just 200 rounds, the average lowest cost attained using the interaction table is 69.2 while the average lowest cost without using the interaction table is 93.9. Interaction produces a cost savings of approximately 26%. Some of this benefit is realized from eliminating conflict-

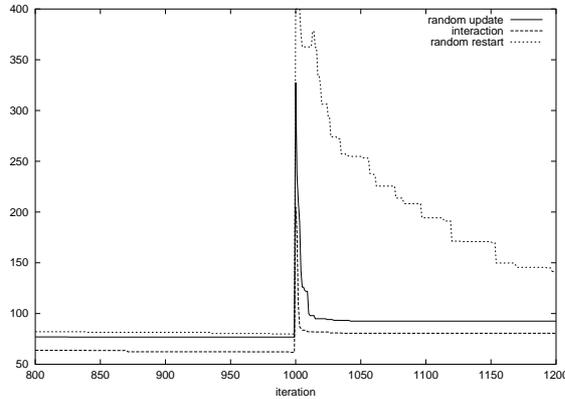


Figure 2.15: Results for dynamic mediation.

ing proposals (9%); the remainder comes from the probability distribution that weighs positive and negative interactions when assigning agents to tasks in a proposal. The results confirm our intuition that in certain domains rich bids and the interaction table benefit task assignment.

The goal of the second set of simulations was to validate the use of rich bids and the interaction table dynamic changes occur in the environment. The expectation was that since interaction information was stored for task types, using that information when a new task appeared would allow better task assignments to be made.

Figure 2.5.2 illustrates the average lowest cost assignment found before and after two new tasks are introduced at round 1000 of the mediation. Three different update procedures were compared: random update, interaction, and random restart. Interaction and random update proceed similarly to interaction and random in the first simulation for the first 1000 runs. When new tasks appear, these procedures use the best assignment found so far for the old tasks and assign new agents to the new tasks. Random update assigns agent randomly for the new tasks; interaction uses the task type interaction information previously reported to assign the new tasks. *Random restart* ignores the best task found so far and randomly assigns agent to all of the tasks.

As expected, there is a benefit in average lowest cost to starting with the best assignment found so far, as random update and interaction perform better than random restart for all of the 200 runs shown after the new tasks appear. In addition, interaction performs better than random update which indicates that information from rich bids recorded before the new tasks appear is useful when assigning the

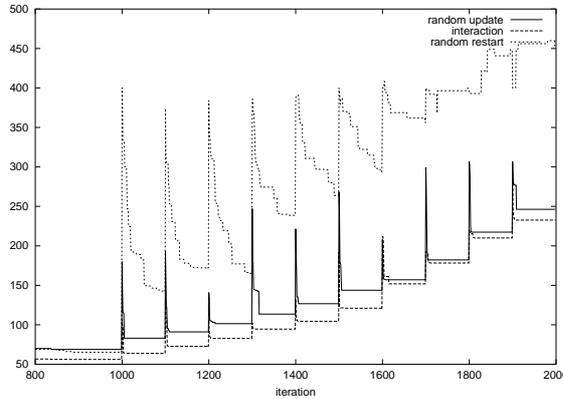


Figure 2.16: Results for many task appearances.

new tasks. The results shown in Figure 2.16 further strengthen this finding: the interaction continues to perform well when a single new task appears after every 100 rounds in negotiation after the first 1000 rounds.

2.6 System architecture: interleaving negotiation and execution

The implementation of the ideas described in this chapter within the context of the challenge problem requires agents that do more than just bid on tasks and take measurements. The system was architected so that agents would normally take on what we will refer to as *background team commitments*. These are commitments that are not related to the result of an award from a negotiation. We found that establishing and honoring such team level commitments played a central role in producing responsive behavior. In the sensor problem domain, the background commitments can involve, for example, opportunistically responding to measurements taken while scanning an area. An agent should work for the good of the team and if it accidentally takes a measurement in an area that is part of an ongoing task, it should contribute that measurement to the mediator rather than discard it. We discuss this further below.

In addition, the system was designed to compute expectations or projections of likely future tasks and also to monitor task assignments for progress. Figure 2.17 illustrates the system architecture. The process begins at the top of the figure with groups of agents systematically scanning areas to detect movements. At that

stage, the following actions are performed:

1. All sensors turn sectors on in sequence (no assumptions are made about possible starting points)
2. Each sensor that detects a target broadcasts to the group
3. A tracker agent (either pre-designated or the agent with the lowest id) estimates location based on knowledge of other known targets
4. The tracker agent outputs likely sectors based on scanning geometry (See the next section)

Once a target is detected, the sector, S , associated with it is output to a process which forms an initial team tasked to triangulate and estimate the target's location and velocity. The sector, S , represents a crude measurement of its general location. The input to that stage is an estimated location in terms of S . The steps performed at this stage are:

1. Compute sectors, S' , that overlap with S
2. Assign nodes associated with S' to aim sensors at S and take frequency and amplitude measurements
3. Compute and output location and velocity estimates

One of the agents in the team is then chosen as the mediator and projects that target forward in time. The projected path is then segmented into a set of tasks, $projected_track = \{(L_1, T_1), (L_2, T_2), \dots\}$, where each task is represented as a pair (L_i, T_i) of location and time, respectively. A candidate team is then chosen based on its proximity to $projected_track$. The project path is then input to the allocation process (mediation or auction) which computes an initial allocation and, if there is more time, conducts additional rounds of mediation to improve that allocation. At that time, background commitments are also identified. Once it becomes time to act, the allocation or set of commitments is communicated to the tracker (in our system, some distinguished agent(s) responsible for this task) which then begins collecting measurements from the allocated team members and computing a track for the target. Finally, the monitor process compares the projected and the actual tracks and re-starts the entire process depending on the result.

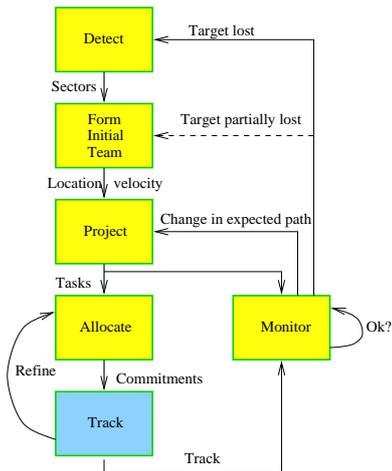
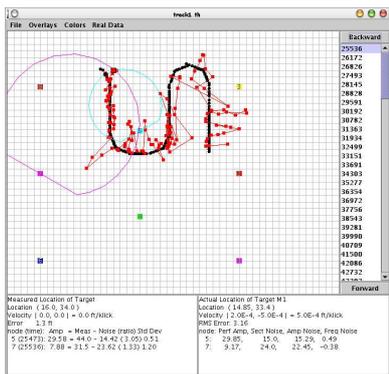
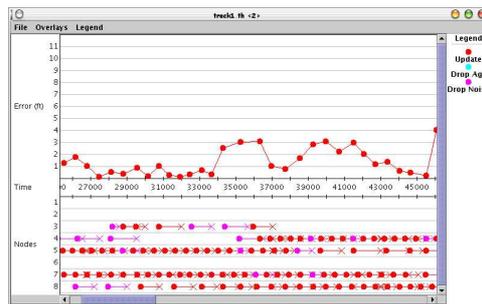


Figure 2.17: Architecture



a) UMass visualization



b) Time line showing sensor activation

Figure 2.18: The UMass visualization tool

2.6.1 Visualization tools and geometric reasoning

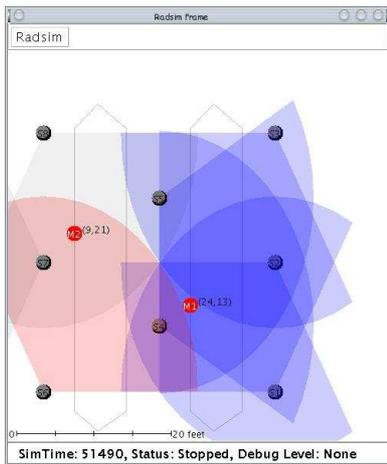
One aspect of the evaluation that we conducted of our system took place during development time. We made extensive use of a visualization tool developed by the University of Massachusetts (UMASS) (See Figure 2.18). Using this tool, we were able to ascertain whether the sequences of measurements taken by agents were synchronized; the knowledge gained often enabled us to more easily locate problems with the algorithms we had developed. For example, if the visualization tool detected that an agent was taking less than one measurement per second during tracking, it was likely that there was a problem with the algorithms that implemented the system architecture.

To help distinguish those problems that were associated with the tracker and those that were associated with the negotiation algorithms, we developed a *geometric tracker*. This tracker combines a conservative sensor coverage model together with information it receives indicating that an object is moving *somewhere* in a sector. By overlapping several zones that represent the areas sensed by the sensors (see Figures 2.19 a,b), one can determine the general area in which the target is located. By adding more sensors and more measurements, the area of uncertainty becomes smaller. The purpose of the geometric tracker was to determine if the tracker's result was actually plausible. For example, in Figures 2.20 e and f, the tracker is reporting an incorrect position for the the target. Without the geometry tracking, the agents would have redirected their sectors to the new position where there was in fact no target, thus causing inefficiency by wasting resources.

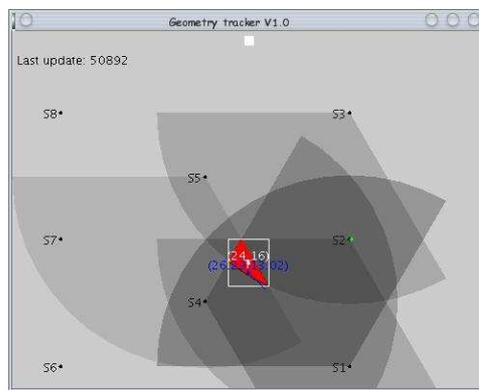
This tracker was effective for two reasons. First, the sensors did an excellent job in detecting movement; the number of false-positives (i.e., the number of detect movements when they was actually none) was almost zero. Second, the placement of the sensors was done intelligently to maximize coverage making the overlapping area small.

In Figure 2.19-d, the four vertically aligned small dots within the box are the segments that were actually auctioned off during the next ten second period. Those segments were based on the projection (darker line in Figure 2.19-d just above those dots) of the target direction.

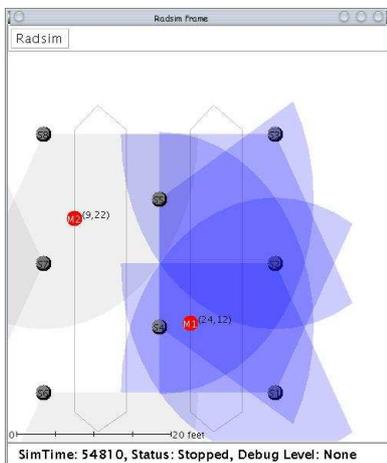
As an example of the sorts of background commitments that were enforced by the system, Figure 2.21 shows a screen capture where node 5 is not part of the team of nodes involved in the current tracking activity; however, it turns out that node 5 happens to “illuminate” the sector where the target is located as part of its normal scanning activity. Rather than discard the resulting measurement, it



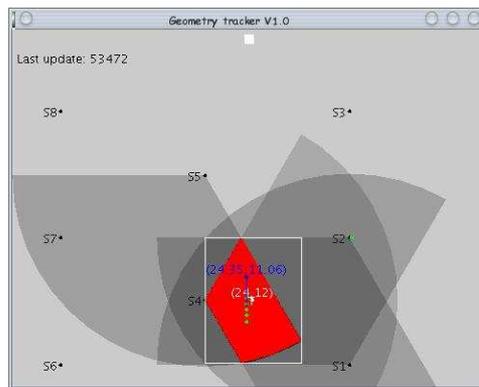
a) Simulator view



b) Geometry tracker

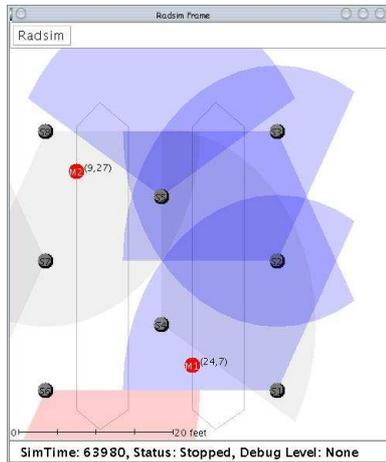


c) Simulator view

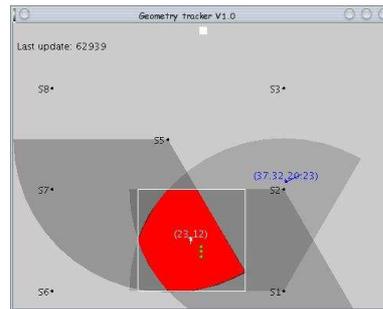


d) Geometry tracker with auctioned points

Figure 2.19: The geometry tracker in action.



e) Bad tracker estimate



f) Bad tracker estimate

Figure 2.20: Identifying bad tracker estimates.

is instead integrated opportunistically to improve the target estimate. In this way, agents would not discard data that could prove useful to the team, even if they were not explicitly asked to collect such data. In some cases, we obtained up to 13% more data points for a track as a result of this addition.

2.6.2 Experimental results

Data was collected from a set of 18 experiments, where each experiment represented an average of three trials, for a total of 54 runs of Radsim using different settings. Each run lasted 240 simulated seconds, while varying various parameters. The purpose of the experiments was to gather data regarding the performance of the system in terms of various parameters such as projection length, number of targets and communication overhead.

2.6.3 Auction results

The first set of experiments involved sequential auctioning in a 16-node sensor arrangement, tracking varying numbers of targets traveling at 0.5 feet/second. The values in each column of the table in Figure 2.22 represent average Root-Mean-Square (RMS) for the given number of targets, and then the average number of messages received by each node for the given number of targets.

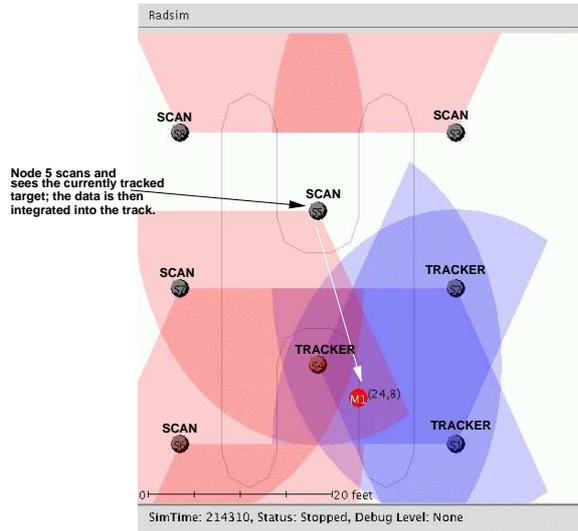


Figure 2.21: Opportunism

RMS	Projection	1 target	2 targets	3 targets	Msgs/Node 1 target	Msgs/node 2 targets
(reactive)	0 sec.	1.94	3.91	24.53	392.4	
	5 sec.	1.25	3.24	26.59	180.01	
(standard)	10 sec.	1.17	3.2	21.45	82.7	192.26
	15 sec.	4.22	7.62	23.82	34.81	

Figure 2.22: Auctioning, 16 nodes, .5 feet per second, 240-second run.

Each row represents a variation in the length of the track projection governed by each auction. The “standard” length during design of the system was 10 seconds, meaning that the auctioneer projected the area to be covered by the target during the ten seconds prior to auctioning tasks in that area. In addition to 5 and 15 second entries, there are also entries corresponding to experiments involving projections of length 0, which would constitute a purely reactive auctioneer. That is, such an auctioneer would continuously instruct the other nodes to immediately fire at the current point.

The second set of experiments utilized the same target speed of 0.5 feet/second, and focused solely on a 10-second projection length. The system consisted of a 24-node configuration. The results are shown in Table 2.23.

Projection	1 target	2 targets	3 targets
10 sec.	1.24	3.12	24.58

Figure 2.23: Auctioning, 24 nodes, .5 feet per second, 240-second run.

RMS by Projection Length Figure 2.24 illustrates a fairly uniform degradation in track quality as the number of targets increased, regardless of the chosen projection length (using sequential auctions). That is, RMS error increased as the nodes must track more targets, making the largest jump with the addition of a third target. Specific values are shown in Figures 2.26 through 2.29.

Figure 2.24 indicates that a 10-second projection was ideal for the given auctioning system, always resulting in less error than other settings. As the projection became shorter, to 5 or 0 seconds, slightly more error arose. We found the largest error at a 15 second projection, in all but the three target cases.

When the projection area was too small, agents became overly reactive; we found that there was not enough latency and agents were slightly more likely to turn away from the correct firing sector whenever there was such a momentary fluctuation in the track synthesizer. However, for the same reason, they were also able to recover fairly quickly from a momentary error, as they did not commit to such a reading for as long a period of time.

We found that a long projection length of 15 seconds was better at insulating the system from fluctuations; however, when the system received a bad reading from the tracker, it was more likely to incorrectly maintain its focus on the wrong sector. Specifically, unless the target later deviated by such a margin that the auction correction mechanism would override the current assignment, the nodes would fire upon the incorrect position for a full 15 seconds. This resulted in longer stretches of lost tracks and hence a sharp increase in error in all cases but the three-target one, which was so difficult to track that RMS was less statistically significant.

Messages by Projection Length Figure 2.25 illustrates another consequence of varying projection length, conservation of messages. Here the average number of messages received by a node was compared for each of the four experimental projection settings, during tracking of a single target.

When a projection covers a longer span of time, auctions will be less frequent, resulting in fewer messages within the same 240-second trial. The reason the curve appears more logarithmic than linear is that slowing the frequency of auc-

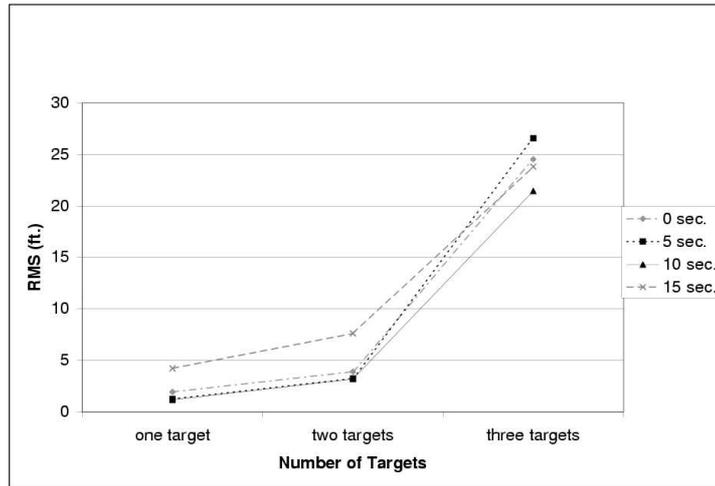


Figure 2.24: RMS by projection length.

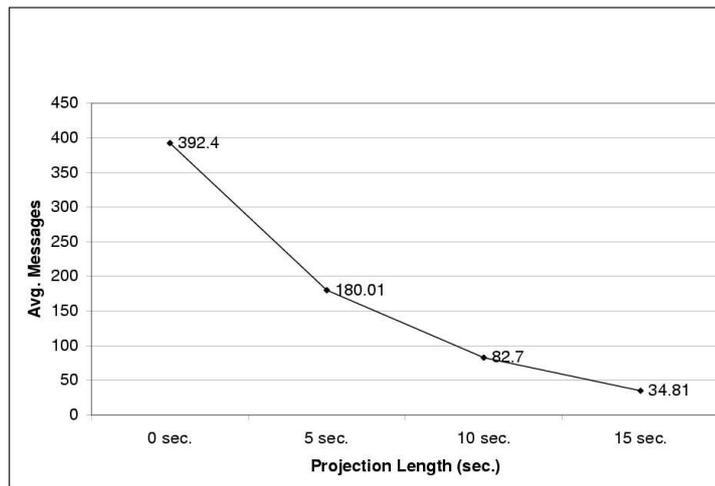


Figure 2.25: Messages by projection length.

tions additionally mitigates the need for re-sends and overrides, which resulted in disproportionately higher message traffic.

The given measurements do not include messages containing measurement reports to the track synthesizer, which varied arbitrarily across projection length settings.

Track Quality Figures 2.26 through 2.29 report results on track quality for 0, 5, 10, and 15 second projections. Figure 2.26 show the performance of the 0-second projection, giving specific values for RMS. Track quality degraded with the addition of targets, especially with the addition of the third target. The track quality degradation resulted from a disproportionate increase in target-differentiation problems: the results were very poor RMS scores.

In particular, the introduction of a third target makes it especially likely that two targets will pass through the same sector for a given node, resulting in a composite reading that is not well-handled by the tracker. In addition, when the two targets exit the area and then diverge, there is no way to match the subsequent tracks with the tracks for the targets as they entered the area. Such a problem exists even if the paths do not include sharp turns, so long as the point of intersection does not fall right on a sector boundary.

Track Quality by Number of Nodes Figure 2.30 compares track quality for various numbers of targets when the number of nodes was increased; these experiments were limited to 10 second projections. In particular, the same target paths were tracked by 24 nodes rather than 16 nodes, resulting in comparable track quality.

This was not surprising in the case of one or two targets, as they were already well-covered by 16 nodes. Significantly, though, there was no improvement in any of the trials for three targets, where the 16 nodes often were not able to devote more than three sectors to a given target. Once again this was attributed to problems with target differentiation.

2.6.4 Mediation experiments

The purpose of the second set of experiments was to evaluate the effect of the choice of group decision-making procedure on the track quality achieved by a group of sensors. We compared auctions to mediation in an environment consisting of 16 nodes using a 10-second projection length. As shown in Figure 2.31,

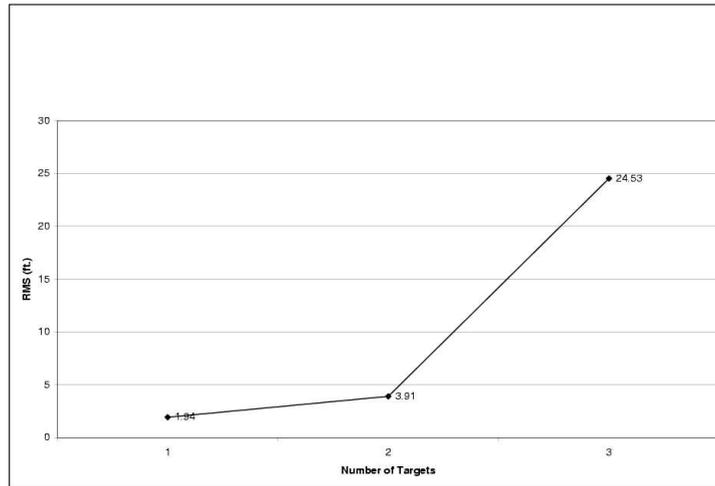


Figure 2.26: Track quality: 0 second projections.

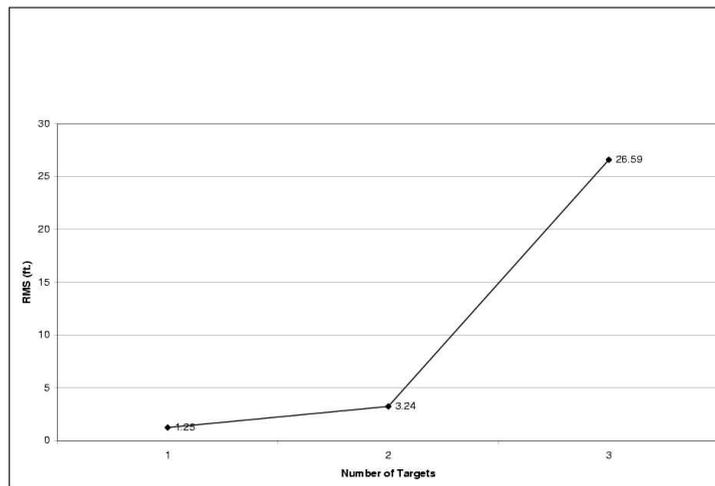


Figure 2.27: Track quality: 5 second projections.

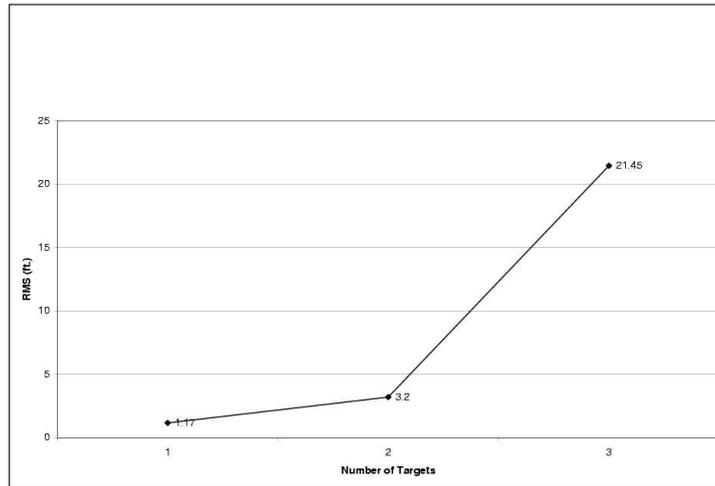


Figure 2.28: Track quality: 10 second projections.

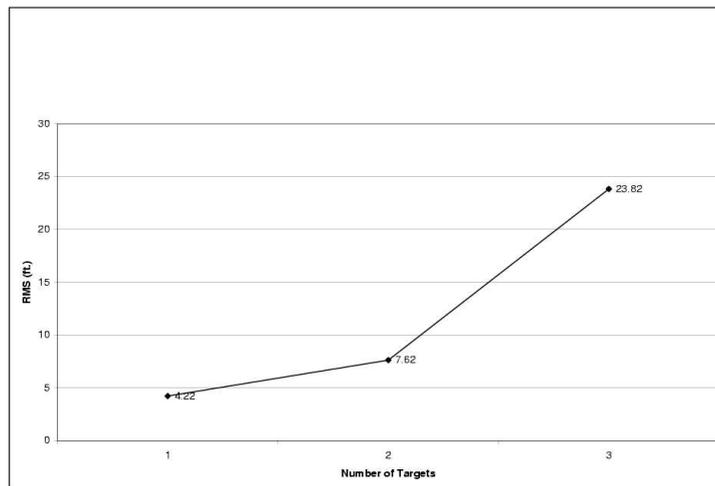


Figure 2.29: Track quality: 15 second projections.

the RMS error achieved by auctions was similar to that achieved by mediation for both the one- and two-target environments.

The main difference observed between the two approaches involved the number of messages passed. The auction used less inter-agent communication messages to achieve its task allocation. In the system, it was the number of messages passed, rather than the length of those messages, that most affected performance. Due to the higher number of messages required by mediation, and the large overhead associated with message passing in the system, mediation was unable to find feasible allocations when targets traveled at the usual 0.5 feet per second. As a result, the experiments for mediation were conducted with a target speed of 0.1 feet per second.

As illustrated in Figure 2.32, the number of messages required for auctions grows with the number of targets (or, group decisions) that must be made. It shows that mediation is able to find an allocation that is as effective as auctions with a constant number of messages. In this domain, auctions are able to find the best allocation with a small number of messages because of both the small size of the search space and the small size of the information set that must be communicated to the auctioneer: there is only a small degree of interaction between tasks, and the suitability of tasks to agents is based on distance from the target which is easy to communicate. The advantage of mediation lies in its simplicity and scalability: the mediator needs no prior information about the domain, and the agents respond to fully specified proposed outcomes. As a result, mediation would extend well to problems with a larger number of targets in this domain. Mediation could allocate tasks in a constant number of messages, while with auctions the number of messages required would increase. Practical limitations in our tracking technology prohibit us from exploring results of experiments with more than two targets.

The last graph compares the number of messages per node utilized by the two approaches. While the auctioning system must send more messages for more auctions with the addition of each target, the mediation system importantly uses a virtually constant amount of communication.

While auctioning better conserves communicative resources for one or two targets, after that it will far surpass mediation in generating messages. Message count is almost constant for mediation because each allocation message governs all targets at once.

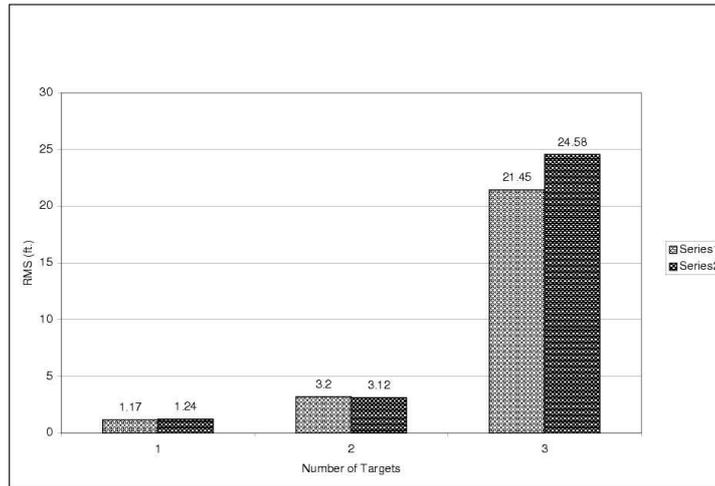


Figure 2.30: Track quality by number of nodes.

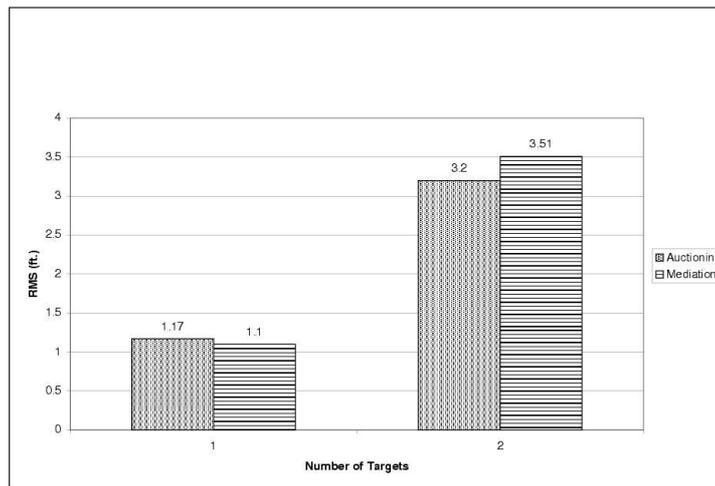


Figure 2.31: Auctioning versus mediation: quality.

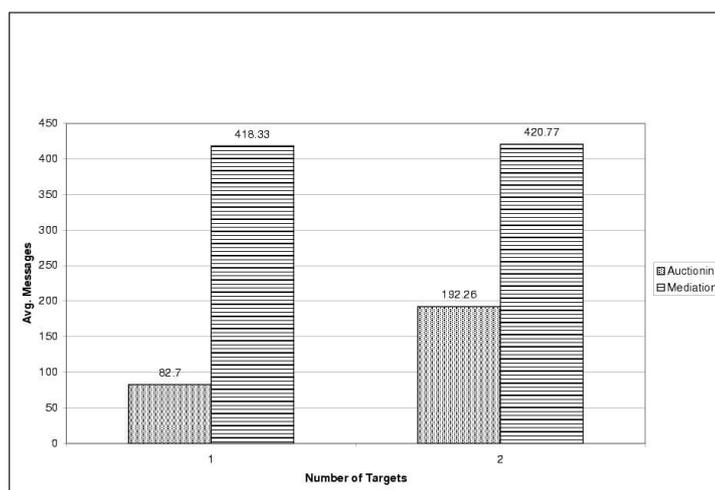


Figure 2.32: Auctioning versus mediation: messages.

2.7 Summary and related work

Research in auction algorithms has generally assumed that agents are self-centered. Such an assumption is appropriate for allocation of items such as tobacco or flowers, where an agent’s well-being is only affected by those items that it is or is not allocated. However, as we have discussed, in multi-agent task allocation situations one agent may feel strongly about task allocations to other agents in the group.

Historically, task interaction has been cited as an additional problem with using auctions for task allocation. Combinatorial auctions [[Sandholm 1999a, Hunsberger and Grosz 2000, inter alia]] provide a solution to this problem by allowing agents to place “exclusive or” bids on bundles of tasks: an agent can bid accurately on sets of tasks that exhibit positive or negative interaction. In addition, each agent may have relevant information as to which other agents are best suited to perform which tasks. In auctions, an agent is assumed to be concerned with only the items it expects to win.

Still, there are challenges in applying combinatorial auctions to problems in which the amount of time available for negotiation is not known in advance. Traditional combinatorial auctions require a two step process where first agents gen-

erate and submit bids, and then apply a winner determination algorithm to determine the best outcome. While the winner determination phase may be anytime, the bid generation and group context problems that we have discussed still remain. A related problem with combinatorial auctions is completeness: all tasks need to be allocated. A complete allocation may not be possible when each agent is generating bids independently.

Iterative combinatorial auctions [[Parkes 1001]] have been proposed as an alternative that simplifies the bid generation phase by allowing agents to reveal their bids incrementally. Agents with complex utility functions in environments with high costs of communicating them may substantially benefit from such a mechanism.

Task re-allocation has been addressed by the literature in contract net protocols [[Smith and Davis 1983, Sandholm 1998]]; in mediation, task re-allocations are considered during each new cycle. Sandholm [[Sandholm 1998]] proves that the optimal allocation may only be found by allowing arbitrarily complex contracts (of type OCSM) to be made; these contract can be so complex that they may involve all the agents in the group, thereby eliminating the advantage of a decentralized approach. The Allocation Improvement algorithm described in Figure 2.5 extends this work to provide a concrete implementation of an algorithm for the ordering successive OCSM contract exchanges. In addition, the work described in [[Sandholm 1998]] does not address the problem of dynamic response to changes in the environment.

The major difference in the assumptions made in work on contract nets and in the work reported in this chapter is in the incentives of agents. Sandholm's work assumes that an agent's decision making is based on myopic best response and requires inter-agent monetary payments to induce task reallocation. The agents described in this chapter are assumed to be interested in maximizing the objective function, and as a result do not require monetary payments as an incentive to exchange tasks.

Game theoretic models involve analysis and development of systems for multi-agent interaction. That work focuses on ways that agents in a group make decisions when the results of those decisions co-depend on the decisions of others in the group. Mechanism design [[Parkes 1001, inter alia]] involves developing algorithms aimed at ensuring agents truthfully reveal relevant information about their preferences. Under our assumption that the goal of the agents is to maximize the objective function, agents will not be willing to incur a cost to ensure truthful revelation.

Work in contract nets does not fall squarely into the area of game theory be-

cause agents are assumed to exhibit myopic best response strategies rather than omniscient self-interest. The assumption made in this chapter of an incentive to maximize the group objective function is yet another approach. The decision of which approach is more appropriate depends on the domain and on the qualities of the agents that are negotiating. Study of agents with bounded rationality [[Sandholm 1999b, inter alia]] promises to provide insight into making this decision.

Research in Constraint Satisfaction Problems (CSPs) has addressed issues related to those we have discussed with respect to dynamic mediation (see, for example, Chapter 10 of [Lesser et al 2003]). These include adapting a CSP solution to changes in the environment when those changes are expressed as new constraints[[Dechter 1988, Verfaillie and Schiex 1994]]. Algorithms for Distributed CSP (DCSP) [[Yokoo and Ishida 1999]] distribute relevant constraint information among several agents. Heuristic CSP repair methods have been explored in the context of dynamic rescheduling [[Minton *et al* 1992]]. One important difference with dynamic mediation is that in CSPs any variable assignment that satisfies the problem's constraints is a solution; there is no welfare function to be optimized.⁴ In addition, DCSPs only support communication of negative interactions in terms of *nogoods*.

The MARS system [[Fischer *et al* 1995]] addresses dynamic multi-agent renegotiation focusing on replanning within a single agent and requiring communication with other agents only when a new single-agent plan cannot be found. Dynamic mediation supports negotiation among *all* agents.

The focused D* algorithm is a real-time replanning algorithm that has been applied to robot path planning in partially known environments [[Stentz 1995]]; the arc costs in the search graph can change during the search process. It has been suggested that negotiation can be viewed as a form of distributed search [[Durfee 1999, Ephrati and Rosenschein 1993]], it would be interesting to cast negotiation first as a distributed search problem and then apply an algorithm such as focused D*. Research in self-stabilizing algorithms focuses on algorithms that adapt to transitions to *any* arbitrary state [[Dolev 2000]]; instead, we have focused on particular types of faults.

In summary, we have explored the class of center-based algorithms and, in particular, introduced a new algorithm, called Dynamic Mediation, through which teams of cooperative agents can negotiate over the distribution of tasks while allowing solutions to be adapted to a dynamically changing environment in which

⁴However, see the work described in Chapters 10, 11 and 13 of [Lesser et al 2003].

new tasks and agents can appear or faults can occur. The algorithm makes use of prior progress without requiring that the entire negotiation be restarted. In Dynamic Mediation an agent's bid need not be restricted to a single value for a task but can include additional useful information in the form of potential positive and negative interactions with other tasks. Rather than communicate such interaction information in terms of fully specified and executable tasks, agents maintain task type information which is more general and can be used by the mediator to prune the negotiation space. We have also presented a new algorithm, together with promising results in a synthetic domain, that addresses the combinatorial bid generation problem. Finally, we have presented a system architecture that addresses execution-time issues involving monitoring, re-negotiating and establishing background team commitments.

We have described experimental results in various domains including the distributed sensor challenge problem. The experimental results suggest that dynamic negotiation methods have significant promise. Based on our experiments, we conclude that it is best to use dynamic mediation when time is an important resource and negotiations must end quickly. If negotiation time was unlimited, the quality of outcomes attained by static methods would eventually catch up to those attained by dynamic methods. We are currently experimenting with extensions that relax the restriction to non inter-agent dependencies in proposals and that involve the use of a general task abstraction hierarchy to the bid generation problem.

Chapter 3

Resource allocation in very large-scale environments

3.1 The Distributed Dispatcher Manager

In this chapter we consider the complexities that arise when one scales up distributed agent networks to thousands of sensor agents and thousands of objects. We describe a system for effectively managing such networks, called the Distributed Dispatcher Manager (DDM). DDM differs in a number of important ways from other systems.

Mobility: We focus on mobile sensor agents; agents remain relatively simple and lightweight.

Organization: The complexity of the distributed control problem for such massive agent systems is managed through a hierarchical organization in which teams of agents are associated with sectors; teams themselves can represent elements of other teams.

Tracking: We have extended the tracking algorithms discussed so far in this book so that a *single* agent can track an object by taking multiple sequential measurements and combining them. We assume that multiple objects can be discriminated within the field of a sensor. Finally, we do not focus on the tracking of a *particular* object, but rather on adequate coverage of given areas.

Task synchronization: One consequence of the above extensions to the tracking

algorithm is that the communication requirements between agents are lessened and, in particular, synchronization between agents is not necessary.

Sensors: DDM, in its current state, does not manage the usage of certain resources, such as sensor power utilization. In addition, our treatment of sensor noise is a bit different from the systems described so far in that, in DDM, some measurements are lost, but the ones that are not lost are accurate.

Simulation: In order to experiment with DDM in domains of the above sort, we have developed a simulation which reflects the above assumptions. In some cases, the set of environments constructed for testing the system have been complicated to reflect such complexities; in other cases, a number of simplifying assumptions have been made so as to be able to focus on the scale-up issues (for example, we focus on objects which move in a straight lines).

DDM organizes the Dopplers in teams, each with a distinguished team leader. A team is assigned a specific geographic sector of interest. Each Doppler can act autonomously within its assigned area while processing local data. Teams are themselves grouped into larger teams. Communication is restricted to flow only between an agent (or team) and its team leader. Each team leader is provided with an algorithm to integrate information obtained from its team members.

We present results from experiments that involved hundreds of agents and more than a thousand objects. These results support our hypothesis that DDM is successful in large-scale environments. The experiments also allow us to examine the question of how to determine the number of levels of the DDM hierarchy in a large-scale system. Our results show that as the number of levels of the hierarchy increases the quality of the results slightly decreases. However, the time complexity of the system decreases exponentially. Consequently, we found that using too few levels may not suffice to solve the global problem.

In the next section we describe the large scale ANTS problem and present the main elements of DDM. Subsequently, we detail the comprehensive study we conducted to evaluate the hierarchical solution. We conclude by discussing the major contribution of our solution to the large-scale agent system challenges in terms of capability, accuracy, efficiency, cost-effectiveness, robustness and fault tolerance.

3.2 The large scale ANTS challenge problem and the DDM

We consider a large-scale environment where there are many mobile targets and many mobile Dopplers moving in a specified geographic area. The goal of the DDM system is to track the targets. Each target moves at a steady velocity along a straight line. Targets differ from each other by their motion properties. Motion properties define the target state, location and velocity, at any given time. Both location and velocity are vectors. The location vector is referred to in the physics literature as the radius vector, the vector from the axis origin (0,0) to a target. The velocity vector describes the change in a target's location every second. A steady motion equation may look like the following [[Feynman 1963]]:

$$f(t) = \langle \bar{r}_0 + \bar{v} \cdot t, \bar{v} \rangle$$

where \bar{r}_0 is the location of the target at time $t = 0$ and \bar{v} is the velocity vector. The goal of the DDM system is to identify the motion equation of each target in the area.

DDM uses agents to find the set of motion equations that represent the targets. We will refer to this set as the global information map, denoted by *InfoMap*. The base level of the DDM consists of mobile sampling agents. Mobile sampling agents are agents that use simple Doppler sensors to sense targets. Each of these sampling agents moves autonomously according to a predefined movement algorithm. Each agent periodically stops briefly to take measurements. When an agent takes measurements we refer to its state as the *viewpoint* from which a particular object state was measured. The measurements alone are not sufficient to identify the exact state of the observed target. The sampler agent can only determine two possible states of the observed target based on four consecutive measurements. Only one of them is the correct state of the target at the observation time. Using such estimates, a sampler agent then produces what we call a *capsule*. A capsule consists of (a) two possible states of a target, (b) the time associated with the measurements that were used to infer the possible target states, and (c) the state of the agent, i.e., its location and orientation during the measurements. Later on we will show how sampling agents transform measurements into capsules.

The DDM's goal is to estimate the motion equations of the targets using capsules. The equations can be deduced from a sequence of target states. The main problem faced by the DDM with respect to a capsule is how to choose the correct state. To resolve this problem, the DDM makes use of the fact that two states

may be the correct states of the same target if they are instances of the same motion equation. We introduce a relation called *ResBy* that holds if the state of one target comes about from another. DDM tries to match states of capsules using this relation and makes linked lists of target states. Each linked list represents a potential target movement. We refer to this linked list of target states as a *path*. DDM also attempts to determine the accuracy of potential paths. It assumes that if two or more target states in a path were recorded by different agents, then the path represents target motion accurately.

In a large-scale environment, DDM would have to link many capsules from the entire area of interest. Applying the relation *ResBy* many times is time consuming. However, there is a low probability that capsules created based on measurements taken far away from one another will fit. Therefore, the solution is distributed. The DDM uses hierarchical structures to construct a global *infoMap* distributively. The lower level of the hierarchy consists of the sampling agents. These agents are grouped according to their location. Each group has a leader. The sampling agents create capsules and send them to their group leaders. The second level of the hierarchy consists of the sampler group leaders. Each sampler group leader obtains capsules only from the sampling agents in its group. This limits the time that it needs to process the capsules, but may reduce its ability to link between states since it obtains only a portion of the capsules.

The sampler leaders are also grouped according to their areas of responsibility. Each such group of sampler leaders is associated with a zone group leader. A sampler leader sends its zone leader its estimates of target motion equations in its area and capsules that it was not able to use in the estimation process. The third and higher levels of the hierarchy consist of zone group leaders, which in turn, are also grouped according to their areas of interest. Zone leader agents are responsible for retrieving and combining information from their group of agents. That is, they try to estimate the motion equations based on the estimates they received from agents in their zone. All communication is restricted to exchanges between a group member and its leader. The information unit sent by leaders to their higher-level leaders is called a *local information map*. A local information map, which we refer to as *localInfo*, is a triple consisting of: (i) an accurate solution component consisting of a set of motion equations that with a high probability represent targets; (ii) a semi-accurate solution component consisting of a set of paths; and (iii) a set of capsules that were not used for the formation of any motion equation of (i) or any path of (ii). That is, each zone leader obtains local information maps of all its agents and combines them into an information map of its area. Thus, the top-level leader agent forms a local information map of the entire area.

To conclude, the formation of a global information map integrates the following processes:

- Each sampling agent gathers raw sensed data and generates capsules.
- Every dT seconds each sampler group leader obtains from all its sampling agents their capsules and integrates them into its *localInfo*.
- Every dT seconds each zone group leader obtains from all its subordinate group leaders their *localInfo* and integrates them into its own *localInfo*.
- As a result, the top-level group leader *localInfo* contains a global information map.

We have developed several algorithms to implement each process. In the next section we present those algorithms.

3.3 Descriptions of algorithms

The first algorithm describes the method for constructing a capsule from raw sensed data. This algorithm is activated by each sampler agent and uses consecutive raw sensed data. The second and the third algorithms describe the way in which every group leader processes incoming local information maps of the sub-areas of its zone to generate a more comprehensive local information map of its entire area.

First, we will describe the main data structures that the agents use. In these data structures specifications and in the algorithm descriptions, we will use a dot notation to describe a field in a structure, e.g., $c.sa$ is the sampling agent field of the capsule c .

Target state: $s = \langle \overline{D}, \overline{V} \rangle$ where D is the location of the target and \overline{V} is its velocity at a given time. For example: $\langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$ is a target state where the target was at location $X = 100$ and $Y = 100$ and its velocity was $V_x = 2$ and $V_y = -1$.

Sensing agent state: $sa = \langle \overline{D}, O \rangle$ where \overline{D} is the location of the sensor and O is the orientation of the sensor. For example: $\langle \langle 150, 150 \rangle, \pi/2 \rangle$ is an agent's state where the sensing agent was at location $X = 150$ and $Y = 150$ and had an orientation of $\pi/2$.

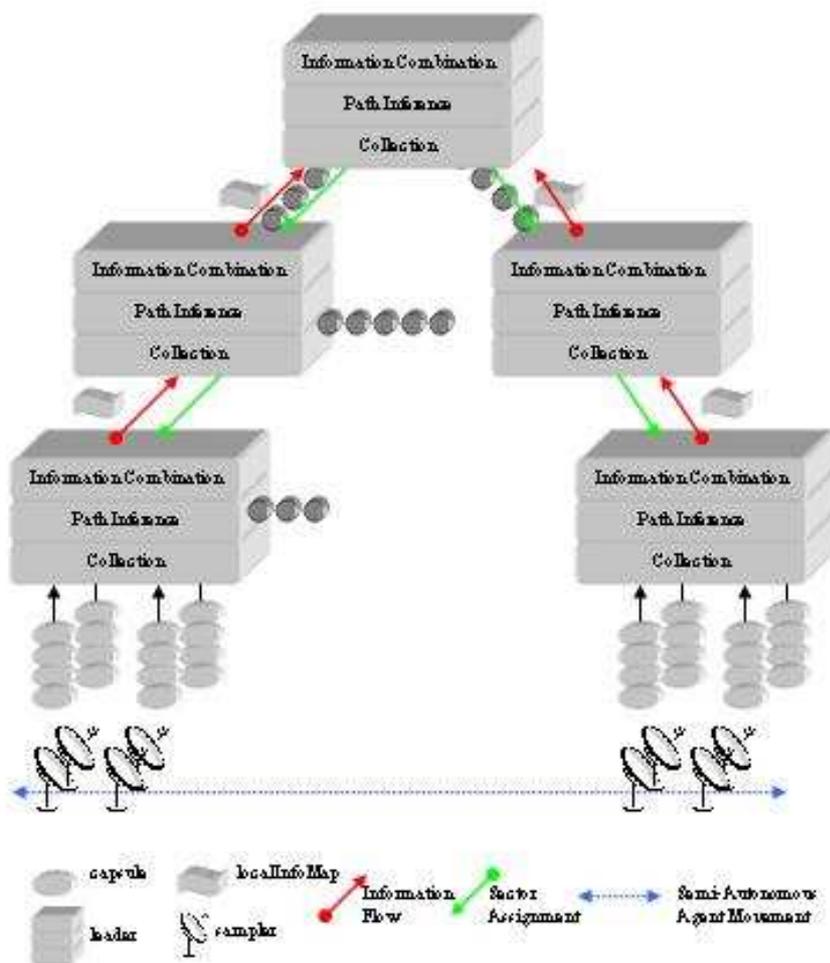


Figure 3.1: DDM hierarchy information flow diagram.

Capsule: $c = \langle t, sa, \{s_1, s_2\} \rangle$ where t is the time of the sampling, sa is the sensing agent's state during the time the measurements that were used for the formation of the capsule's target states were taken, and s_1, s_2 are two possible target states computed by the sampler agent. An example is, $\langle 30, \langle \langle 150, 150 \rangle, \pi/2 \rangle, \{ \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle, \langle \langle 200, 200 \rangle, \langle -2, -3 \rangle \rangle \} \rangle$ where the time of the sampling was 30 and the sensing agent state was $\langle \langle 150, 150 \rangle, \pi/2 \rangle$ where the two possible target states were $\langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$, and $\langle \langle 200, 200 \rangle, \langle -2, -3 \rangle \rangle$.

Path point: $\pi_i = \langle t_i, sa_i, s_i \rangle$ where t_i is the time of the point, sa_i is the sensing agent's state during the time the measurements that were used to compute the point's state were taken, and s_i is the target state of the point. For example: $\langle 30, \langle \langle 150, 150 \rangle, \pi/2 \rangle, \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle \rangle$ where while the time of the path point was 30, the sensing agent state was $\langle \langle 150, 150 \rangle, \pi/2 \rangle$ and the target state was $\langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$.

That is, while in a capsule there are two possible states associated with measurements, in a path point there is only one. The goal of the agents is to choose the correct one.

Path: $p = \langle \pi_1 \dots \pi_n \rangle$ where π_1 and π_n are the first and the last path points. Every pair of path points in a path satisfies the ResBy relation.

Target state function: $f_{\pi_s, \pi_e}(t) = \langle \pi_s.s_s.\overline{D} + \pi_s.s_s.\overline{V} \cdot (t - \pi_s.t, t), \pi_s.s_s.\overline{V} \rangle$ valid in the range $\pi_s.t \dots \pi_e.t$. For example: if

$\pi_s = \langle 30, \langle \langle 150, 150 \rangle, /pi/2 \rangle, \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle \rangle$ and

$\pi_e = \langle 40, \langle \langle 450, 25 \rangle, /pi/4 \rangle, \langle \langle 120, 90 \rangle, \langle 2, -1 \rangle \rangle \rangle$ then $f_{\pi_s, \pi_e}(t) = \langle \langle 100, 100 \rangle + \langle 2, -1 \rangle \cdot (t - 30), \langle 2, -1 \rangle \rangle$ and at $t=30$ we have that

$f_{\pi_s, \pi_e}(30) = \langle \langle 100, 100 \rangle, \langle 2, -1 \rangle \rangle$ and at $t=40$ we will have that,

$f_{\pi_s, \pi_e}(40) = \langle \langle 120, 90 \rangle, \langle 2, -1 \rangle \rangle$. For simplicity, we will refer to this function as $f(t) = \langle \overline{D}(t), \overline{V} \rangle$ and to its properties: $f.\overline{D}(t), f.\overline{V}(t), f.t_s$ and $f.t_e$.

Local information map: $\langle \langle f^1, \dots, f^h \rangle, \langle p_i, \dots, p_l \rangle, \langle c_i, \dots, c_m \rangle \rangle$. To form a local information map out of raw sensed data agents should follow a set of steps corresponding to the following stages of data evolution: (i) measurements, (ii) capsules, (iii) path, (iv) target state function and (v) local information map.

3.3.1 The raw data transformation and capsule generation algorithm

The process proceeds as follows. A sampling agent deduces a set of possible target states at a given time to form a capsule. A sampling agent accomplishes this by first taking four consecutive measurements and then creating a new capsule, c , such that the time associated with that capsule corresponds to the time of the last measurement. The state of the sampling agent while taking the measurements is given by $c.sa$. The target states resulting from the application of the function *raw-DataTransformation* to the four consecutive measurements is assigned to $c.states$. The agent stores the capsules until it is time to send them to its group leader. After delivering the capsules to the group leader the sampler agent deletes them. We will now show how an agent transforms four consecutive measurements into a capsule.

A measurement is a pair of amplitude, η , and radial velocity, v_r , values for each sensed target. A radial velocity is the velocity of a target towards the measuring Doppler. Given a measurement from a Doppler radar the target location can be computed using the following equation:

$$R_i^2 = \frac{k \cdot e^{-\frac{(\theta_i - \beta)^2}{\sigma}}}{\eta_i} \quad (3.1)$$

where, for each sensed target, i , R_i is the distance between the sensor and i ; θ_i is the angle between the sensor and i ; θ_i is the measured amplitude of i ; β is the sensor beam angle; and k and σ are characteristics of the sensors and influence the shape of the sensor detecting area. It is possible to infer the exact location of a target by intersecting three different measurements taken at the same time by three different Dopplers. Using the intersection method is very problematic in large scale systems as it requires full synchronization and cooperation between groups of three Dopplers. Thus, the DDM uses measurements from only one Doppler to deduce a possible target state.

It is known that if the location of an object at time 0 is \overline{D}_0 and its velocity is \overline{V} then the next location, \overline{D}_1 , at time 1 of the object is given by:

$$\overline{D}_1 = \overline{D}_0 + \int_{t_0}^{t_1} \overline{V} dt \quad (3.2)$$

where \overline{D}_t is the displacement of the object in time t . If we consider the distance from the center of the Doppler we have that

$$R_1 = R_0 + \int_{t_0}^{t_1} V_r dt \quad (3.3)$$

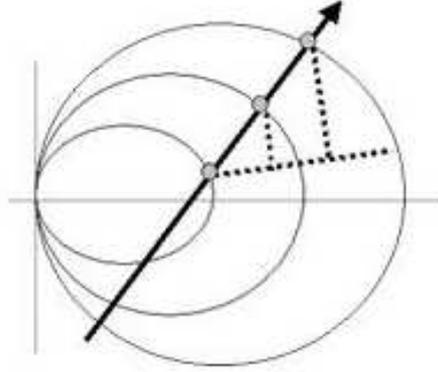


Figure 3.2: Target sampling by one Doppler.

where R_t is the displacement from the center of the sensor at time t and V_r is the relative velocity between the Doppler and the target in the direction of the Doppler's center. We assume that the acceleration of a target over a short period of time is zero. The next target location is therefore:

$$R_1 = R_0 + V_r \cdot (t_1 - t_0) \quad (3.4)$$

We denote $(t_1 - t_0)$ by $t_{1,0}$. From the relation between R, θ and η , given by equation 3.1, we can find the next angle as a function of the former.

In Figure 3.2 the dark arrow represents a target movement vector, the small circles along the target movement represent target locations, $(R_0, \theta_0), (R_1, \theta_1)$ and (R_2, θ_2) , at time t_0, t_1 , and t_2 , respectively, as sensed by the Doppler. Following the projection of R_1 and R_2 over R_0 , and R_{1,R_0} and R_{2,R_0} respectively, as shown by the dotted line, we have the following:

$$\frac{R_{1,R_0} - R_0}{R_1 \cdot \sin(\theta_1 - \theta_0)} = \frac{R_{2,R_0} - R_0}{R_2 \cdot \sin(\theta_2 - \theta_0)} \quad (3.5)$$

Trigonometrically, we may write R_{1,R_0} and R_{2,R_0} as

$$R_{1,R_0} = R_1 \cdot \cos(\theta_1 - \theta_0)$$

and

$$R_{2,R_0} = R_2 \cdot \cos(\theta_2 - \theta_0)$$

by equation 3.1, θ_i can be written as

$$\theta_i = \beta \pm \sqrt{-\sigma \cdot \ln\left(\frac{\eta_i}{k} \cdot R_i^2\right)} \quad (3.6)$$

By substituting R_1 as given by equation 3.4 into equation 3.6 we can deduce that the location, (R_1, θ_1) , at t_1 of a sensed target may be written as a function of θ_0 as specified in the next proposition.

Theorem 3.1 *Assuming that the acceleration of a target in a short time period, $t_{1,0}$, is zero, the next location of the target is then given by*

$$\theta_1(\theta_0) = \beta \pm \sqrt{-\sigma \cdot \ln\left(\frac{\eta_1}{k} \cdot (R_0 + V_{r0} \cdot t_{1,0})^2\right)} \quad (3.7)$$

while

$$R_0 = \sqrt{\frac{k \cdot e^{-\frac{(\theta_0 - \beta)^2}{\sigma}}}{\eta_0}}$$

and

$$R_1 = \sqrt{\frac{k \cdot e^{-\frac{(\theta_1 - \beta)^2}{\sigma}}}{\eta_1}} \quad (3.8)$$

Where $R_0, \theta_0, \eta_0, V_{r0}$ represent values of the target at time $t = 0$ and θ_1, η_1 represent values of the target at time $t = 1$. The same holds for the next angle, θ_2 .

An agent can use the relationship given by equation 3.7 for θ_0, θ_1 and θ_2 together with equation 3.5 to find θ_1 and θ_2 from θ_0 . However, the value of θ_0 is not known and thus can't be used in equations 3.7 and 3.5. Therefore the algorithm examines the range of $0, \dots, 2\pi$ to determine which value of θ_0 solves these two equations.

Note that, given a specific value of θ_0 , the result of equation 3.7 may lead to two valid solutions. This is the reason for the use of capsules: the sampling agent will leave the decision of determining the correct target states to the higher levels.

Equation 3.5 cannot be solved symbolically and therefore the sampler agent uses computational methods. The sampler agent explores the range of θ_0 and looks for suitable locations corresponding to θ_0 . Only certain angles will fit the above equation. To be more precise, the sampler agent uses one more sample and

Find θ_0 function**Input:** $sa, sample_0, sample_1, sample_2$ **Output:** θ_0 minimum_diff= ϵ min_ θ_0 =-1

For $\theta_0 = 0$ to 2π in δ steps
 calculate θ_1 using θ_0 by equation 3.7
 calculate θ_2 using θ_0 by equation 3.7
 diff = the difference between the left side the right side of
 equation 3.5 using θ_1 and θ_2
 if (diff < minimum_diff)
 minimum_diff=diff
 min_ θ_0 = θ_0

Return min_ θ_0 Figure 3.3: Finding a value of θ_0 .

applies the same mechanism to θ_1, θ_2 and θ_3 . Comparing the results from both cases improves the accuracy of the results. The calculated angles are used to form a set of possible pairs of location and velocity of a target (i.e., a capsule). In the algorithms of figures 3.3, 3.4 and 3.5 we will use the notation $sample_i$ to represent a measurement $\langle \eta, V_r \rangle$ at $t = i$.

Theorem 3.2 *The time complexity of the capsule generation algorithm is $O(1)$.***Proof:** While generating a capsule, the *rawDataTransformation* function uses the *Find* function twice. The time complexity of considering the range of all angles from 0 to 2π in *Find* is $O(1)$ as it does the same simple assignments $2\pi / \delta$ times. Therefore, the time complexity of the whole algorithm is $O(1)$.

However, despite the low order, this algorithm can be CPU intensive. A sampler agent applies this algorithm every four consecutive measurements. Thus, it may have to apply it many times if it acquired many measurements or if many targets passed through its sector. A sampler agent may sometimes not have sufficient resources to execute this algorithm many times in real time. In such cases, one can consider using simpler sampling agents, i.e., with smaller detection sectors,

rawDataTransformation function**Input:** $sa, sample_0, sample_1, sample_2, sample_3$ **Output:** *target states* $\theta_0 = \text{Find } \theta_0(sa, sample_0, sample_1, sample_2)$ $\theta_1 = \text{Find } \theta_0(sa, sample_1, sample_2, sample_3)$ if ($\theta_0 \neq -1 \ \&\& \ \theta_1 \neq -1$) $\theta_3 = \text{calculate } \theta_3 \text{ using } \theta_0 \text{ by equation 3.7}$ $\theta_3^* = \text{calculate } \theta_3^* \text{ using } \theta_1 \text{ by equation 3.7}$ if (the difference between θ_3 and $\theta_3^* < \text{epsilon}$) Return $\langle \overline{D}(\theta_3), \overline{V}(\theta_3) \rangle, \langle \overline{D}(-\theta_3), \overline{V}(-\theta_3) \rangle$

else

Return null

Figure 3.4: The rawDataTransformation function

Capsule generation algorithm**Input:** $sa, sample_0, sample_1, sample_2, sample_3$ **Output:** *capsule* $targetStateSet = \text{rawDataTransformation}(sa, sample_0, sample_1, sample_2, sample_3)$ If ($targetStateSet \neq null$) $capsule = \text{new Capsule}()$ $capsule.sa = sa$ $capsule.states = targetStateSet$

else

 $capsule = null$ Return *capsule*

Figure 3.5: Capsule generation algorithm.

which will reduce the computation load on a single agent. One may also consider taking fewer samples.

Example: We will now present an example of how a sampler agent forms a capsule from four consecutive measurements. Consider a case of a sampler agent located at the coordinates $\overline{D}_a = \langle 200, 200 \rangle$ with orientation of 0 degrees. The sampler uses a Doppler with the characteristic $k = 1$ and $\sigma = 1$ and maximum detection range of 100 meters (see Figure 3.1). Consider the following measurements taken by the Doppler.

Time	η	V_r
0	1.08E-04	0.014141
1	1.11E-04	0.042405
2	1.15E-04	0.070619
3	1.18E-04	0.098748

Scanning the range of $0, \dots, 2\pi$ for the value of θ_0 in *Find* θ_0 function, the algorithm computes $\theta_1(\theta_0)$ and $\theta_2(\theta_0)$ using equation 3.7. At the end of the scanning loop the algorithm finds out that while θ_0 had the value of -0.7854 the difference between the left side and the right side of equation 3.5 was minimal. Doing the same in the case of θ_1 , the algorithm finds that when θ_1 was -0.7654 the difference between the left side and the right side of equation 3.5 was minimal.

The algorithm then uses equation 3.5 to find that $\theta_3(\theta_0)$ is -0.72547 and that $\theta_3^*(\theta_1)$ is also -0.72547 . Realizing that the values of the calculated $\theta_3(\theta_0)$ and $\theta_3^*(\theta_1)$ are equal, the algorithm constructs two targets states. The first is $\langle \overline{D}(\theta_3), \overline{V}(\theta_3) \rangle$ and the second is $\langle \overline{D}(-\theta_3), \overline{V}(-\theta_3) \rangle$. The values of $\overline{D}(-\theta_3)$ and $\overline{V}(-\theta_3)$ are given by the following equations:

- $\overline{D}(\theta_3) = \langle R(\theta) \cdot \sin(\theta) + \overline{D}_a.x, R(\theta) \cdot \cos(\theta) + \overline{D}_a.y \rangle$
- $\overline{V}(\theta) = \langle \overline{V}_r \cdot \sin(\theta), \overline{V}_r \cdot \cos(\theta) \rangle$

where $R(\theta)$ is given by equation 3.8 and, in our case, $R(\theta_3) = 70.8378$. According to our example the two target states will be $\langle \langle 153, 253 \rangle \langle 1, 1 \rangle \rangle$ and $\langle \langle 247, 253 \rangle \langle -1, 1 \rangle \rangle$.

3.3.2 Leader localInfo generation algorithm

Every dT seconds each group leader performs the *localInfo* generation algorithm. Each group leader holds its own *localInfo*. The leader starts by purging data older than a predefined τ seconds before processing new data to avoid data overloading.

Updating *localInfo* involves three steps: (i) obtaining new information from the leader's subordinates; (ii) finding new paths; and (iii) merging the new paths into *localInfo*.

Figure 3.6 describes the algorithm for (i) in which every leader obtains information from its subordinates. The sampler group leader obtains information from all of its sampling agents for their *unusedCapsules* and adds them to its *unusedCapsules* set. The zone group leader obtains from its subordinates their *localInfo*. It adds the *unusedCapsules* to its *unusedCapsules* and merges the *infoMap* of that *localInfo* to its own *localInfo*.

Merging of functions is performed both in steps (i) and (iii): this is needed since, as we noted earlier, target state functions that a leader has inserted into the information map are accepted by the system as correct and will not be removed. However, different agents may sense the same target and therefore it may be that different functions coming from different agents will refer to the same target. The agents should recognize such cases and keep only one of these functions in the *infoMap*. We use the next lemma to find and merge identical functions.

Lemma 3.1 *Let $p^1 = \langle \pi_s^1, \dots, \pi_e^1 \rangle$, $p^2 = \langle \pi_s^2, \dots, \pi_e^2 \rangle$ be two paths, where $\pi_j^i = \langle t_j^i, sa_j^i, s_j^i \rangle$ and*

$$f_{\pi_s^1, \pi_e^1}^1(t) = \langle \pi_s^1 \cdot s_s \cdot \bar{D} - \pi_s^1 \cdot s_s \cdot \bar{V} \cdot (t - \pi_s^1 \cdot t), \pi_s^1 \cdot s_s \cdot \bar{V} \rangle$$

$$f_{\pi_s^2, \pi_e^2}^2(t) = \langle \pi_s^2 \cdot s_s \cdot \bar{D} - \pi_s^2 \cdot s_s \cdot \bar{V} \cdot (t - \pi_s^2 \cdot t), \pi_s^2 \cdot s_s \cdot \bar{V} \rangle$$

If $ResBy(\langle t_s^1, s_s^1 \rangle, \langle t_s^2, s_s^2 \rangle)$ then for any $f_{\pi_s^1, \pi_e^1}^1(t)$, $f_{\pi_s^2, \pi_e^2}^2(t)$,

$$f_{\pi_s^1, \pi_e^1}^1(t) = f_{\pi_s^2, \pi_e^2}^2(t)$$

The *mergeFunctions* algorithm shown in Figure 3.6 is based on lemma 3.1. In that algorithm, the leader uses the *ResBy* relation to check whether the first state of the target state function results from the first state of a different target state function. If one of the states results from the other, the leader changes the minimum and the maximum triplets of the target state function. The minimum triplet is the starting triplet that has the lowest time. The maximum triplet is the ending triplet that has the higher time. Intuitively, the two state functions are merged and the new function corresponds to the largest range given the found points. In case that a leader cannot find any target state function to meet the subordinate's function, the leader will add it as a new function to its *infoMap*.

Proposition 3.1 *The time complexity of the obtaining new information algorithm is $O(T^2)$ where T is the number of targets in the τ seconds window of time in which target information is kept by an agent.*

Step 1 - Obtaining new information algorithm

In: LocalInfo

Out: updated localInfo

```
if activated as Samplergroup leader
  for each subordinate sampler, sampler
    additionalCapsules = obtain set of capsules from each sampler
    localInfo.unusedCapsules = localInfo.unusedCapsules  $\cup$ 
      additionalCapsules
else % activated in Zone group leader
  for each subordinate leader, leader
    %in this part we identify identical functions and leave only one of them
    additionalLocalInfo = ask each leader for its local info
    additionalCapsules = additionalLocalInfo.unusedCapsules
    additionalInfoMap = additionalLocalInfo.infoMap
    localInfo.unusedCapsules = localInfo.unusedCapsules  $\cup$ 
      additionalCapsules
    mergedFunctions(localInfo.infoMap, additionalInfoMap);
return infoMap, unusedCapsules
```

Figure 3.6: Obtaining new information algorithm.

mergeFunctions algorithm**In:** target function sets: F, F'**Out:** updated target function: F

```

for each state function,  $f^i$ , in F'
  merged = false
  for each state function,  $f^j$ ,  $f^i \neq f^j$ , in F && not merged
    if ( $f^i.\overline{D}(0) - f^j.\overline{D}(0) < \epsilon\overline{D}$  &&  $f^i.\overline{V} - f^j.\overline{V} < \epsilon\overline{V}$ )
       $f^j.t_s = \min(f^i.t_s, f^j.t_s)$ 
       $f^j.t_e = \max(f^i.t_e, f^j.t_e)$ 
      merged=true
  if (not merged)
     $F = F \cup \{f^i\}$ 

```

Return F

Figure 3.7: mergeFunctions algorithm.

Proof: While obtaining new information, agents implementing the algorithm query each subordinate agent for information. The number of subordinate agents is predefined and therefore constant. In the case of the sampler leader the algorithm combines all capsules. The number of capsules depends on the number of targets up to a constant factor. The constant factor depends on predefined constant values, such as, the number of agents and time period for sampling. Therefore the time complexity for the sampler leader component is $O(T)$. However, for each subordinate leader, the zone group leader also performs the mergeFunctions algorithm. The time complexity of the mergeFunctions algorithm is $O(T^2)$ as it runs over a set of task state functions for every other task state function in another set.

The second step, as shown in Figure 3.10 is conducted by every leader to find paths and extend current paths given a set of capsules. In order to form paths of capsules, the agent should choose only one target state out of each capsule. This constraint is based on the following lemma. According to this lemma one state of one capsule cannot be in a *ResBy* relation with two different states in another capsule, with respect to the capsule's time.

Lemma 3.2 Let $C^1 = \langle t^1, sa^1, \langle s_1^1, s_2^1 \rangle \rangle$, $C^2 = \langle t^2, sa^2, \langle s_1^2, s_2^2 \rangle \rangle$ then if

<i>Sensing Agent State</i>				<i>Target State A</i>		<i>Target State B</i>	
Capsule	Time	Location	Orientation	Location	Velocity	Location	Velocity
0	0	0,0	0	100,100	5,5	60,60	3,-1
1	0	0,0	0	50,50	2,3	30,40	-2,-2
2	1	0,0	0	105,105	5,5	63,59	3,-1
3	1	0,0	0	70,80	2,3	52,53	2,3
4	2	0,0	0	66,58	-1,3	110,110	5,5
5	3	10,10	0	56,59	2,3	30,30	3,3

Figure 3.8: Example of a set of unusedCapsules received by the Finding_new_paths algorithm.

$ResBy(\langle t^1, s_1^1 \rangle, \langle t^2, s_1^2 \rangle)$ then

- (i) $ResBy(\langle t^1, s_1^1 \rangle, \langle t^2, s_2^2 \rangle)$ is false and
- (ii) $ResBy(\langle t^1, s_2^1 \rangle, \langle t^2, s_1^2 \rangle)$ is false.

Proof: If a capsule could be in such a relationship with both target states, the two target states would stand in a ResBy relation between themselves. Given that the two target states have the same creation time and that a target cannot be at the same time in two places, that is not possible.

For the algorithm in Figure 3.10 that creates new paths we add two temporary fields to two of the structures only for the purposes of the algorithm. The first is a boolean flag named *mark* that will be added to the capsule structure. The second is a pointer to the original capsule that will be added to every triple stored in a path. In the first phase, every agent tries to fit every state in unused capsules to an existing path. If the state does not fit, a new path will be created, starting at that state. The second phase the agent separates the paths into accurate and semi-accurate paths according to the number of sampling agents generating them.

Example: Consider a case in which Finding_new_paths algorithm receives the following set of capsules as unusedCapsules:

Let us assume that at the beginning of the algorithm shown in Figure 3.10, allPaths does not contain a path (line 2). Considering each target state in each capsule (lines 3 to 5), we start with TargetStateA of capsule 0. Because allPaths does not contain a path, a new path will be created with TargetStateA of capsule 0 at its head (lines 16 to 18). The next state, TargetStateB of capsule 0, will

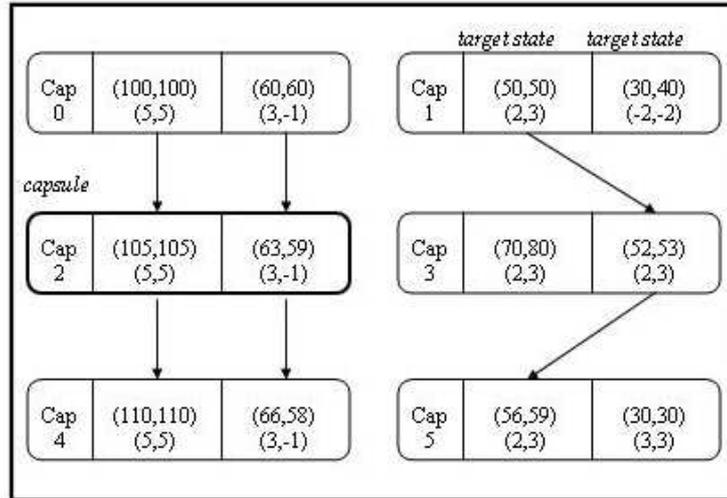


Figure 3.9: An example of an outcome of phase 1

be tested to see if it is in a ResBy relation with any of the tails of the paths, stored in *allPaths* (line 10). It does not stand in a ResBy relation with the only tail that exists: TargetStateA of capsule 0. Therefore, a new path will be created with TargetStateB of capsule 0 at its head (lines 16 to 18). TargetStateA and TargetStateB of capsule 1 will result in a new path as well. However, TargetStateA of capsule 2 is in a ResBy relation with TargetStateA of capsule 0 and will join its path as a new tail (line 14). TargetStateB of capsule 2 will do the same with TargetStateB of capsule 0. At the end of the first phase 6 paths will be formed, 3 of them are made from more than one capsule.

In Figure 3.9 we summarize the outcome of phase 1. Each arrow corresponds to a ResBy relation between two path points. Every capsule contains the original sampler state, which is not shown in the figure due to space limitations.

In phase 2, the algorithm finds that one of the paths in *allPaths* is formed by capsules generated by agents with different states. This path is an accurate path and will be added to the accurate paths set. The algorithm removes all target states that share the same capsule with accurate paths' target states. At the end of the second path, the algorithm looks for semi-accurate paths. A semi-accurate path is a path of target states sensed by the same agent at the same agent state.

The function *pathToFunc* receives a path and returns a function based on it. In

our case it will receive the paths shown in the left side of the figure and return:

$f_{\pi_s, \pi_e}(t) = \langle \langle 50, 50 \rangle + \langle 2, 3 \rangle \cdot t, \langle 2, 3 \rangle \rangle$ with respect to the first and the last path points: $\pi_s = \langle 0, \langle \langle 0, 0 \rangle, 0 \rangle, \langle \langle 50, 50 \rangle, \langle 2, 3 \rangle \rangle \rangle$ and $\pi_e = \langle 3, \langle \langle 0, 10 \rangle, 0 \rangle, \langle \langle 56, 59 \rangle, \langle 2, 3 \rangle \rangle \rangle$.

Proposition 3.2 *The finding new paths algorithm time complexity is $O(T^2)$ where T is the number of targets in the τ seconds window of time passing through the controlled area.*

Proof: In this algorithm, every leader runs over a set of paths for every capsule in its set of capsules. The paths and the capsule sets are correlated with the number of targets in the time period of τ , and therefore results in a time complexity of $O(T^2)$.

3.3.3 The movement of a sampler agent

While the integration algorithms play an important role in producing an accurate infoMap, ultimately, the accuracy of the infoMap fundamentally depends on the accuracy of the observations made by the sampling agents. There are several degrees of freedom associated with the movements of sampler agents. At this point in our research we wanted the sampler agents to move autonomously according to a predefined algorithm without making any assumptions regarding target location. We hypothesize that the following criteria should be considered when determining the sampler agent's behavior: (i) the union of all the sensed area at time t should be maximized and (ii) the intersection of the areas sensed by sampling agent s at time t and at $t+1$ should be minimized. One of the ways to achieve this is to move in the pattern demonstrated in figure 3.11 (page 74). We refer to this pattern as the *Patrol movement pattern*.

We compared the patrol movement pattern with a steady random movement that was used by the agents in [[Yadgar 2002, Yadgar 2003]]. A steady random movement is defined as a movement in a random direction and velocity. Upon reaching the end of the controlled zone, the velocity and the direction is changed and re-directed into the zone. We found out that most of the time the patrol movement pattern is more efficient than the random one. Hence, we will present our simulation results using the former.

Step 2 - Finding new paths algorithm

In: unusedCapsules

Out: updated unusedCapsules, *accurateFunctions*, *mediocrePaths*

% phase 1: make links

(1) sort(unusedCapsules) % by time stamp

(2) *allPaths* = {}

(3) for each capsule, $c = \langle t, sa, \{s_1, s_2\} \rangle$, in unusedCapsules

(4) cap.mark=false % marking for phase 2

(5) for each target state, si , in cap states

(6) linked=false

(7) % because of the above assumption and given that the path

(8) % elements came from capsules there will be only one suitable

(9) % path. Therefore, we exit the loop after finding such a path

(10) for every last triplet, $\langle t_last, sa_last, s_last \rangle$, in each path, p ,

(11) in *allPaths* && not linked

(12) if (*ResBy*($\langle t_last, s_last \rangle$, $\langle t, si \rangle$))

(13) or ($t_last = t \& \& sa_last \neq sa$))

(14) $p = p * \langle t, sa, si \rangle$

(15) linked = true

(16) if (not linked)

(17) $p = \langle t, sa, si \rangle$

(18) *allPaths* = *allPaths* \cup { p }

(19)% phase 2: collect target representing paths that has no common capsules

(20)% when giving a greater priority to paths with more viewpoints

(21) sort(*allPaths*) % by number of viewpoints

(22) *paths* = {}

(23) for each path, p , in *allPaths*

(24) if (not isAnyCapsuleMarked(p) && numberOfViewpoint(p) > 1)

(25) markAllCapsules(p)

(26) unusedCapsules.=unusedCapsules - allCapsules(p)

(27) accurateFunction=accurateFunction

(28) accurateFunctions= accurateFunctions \cup pathToFunc(p)

(29) if activated as top-level leader

(30) mediocrePaths=collectMediocrePaths(*allPaths*)

(31) else

(32) mediocrePaths = {}

(33) Return unusedCapsules, *accurateFunctions*, mediocrePaths

Figure 3.10: Finding new paths algorithm.

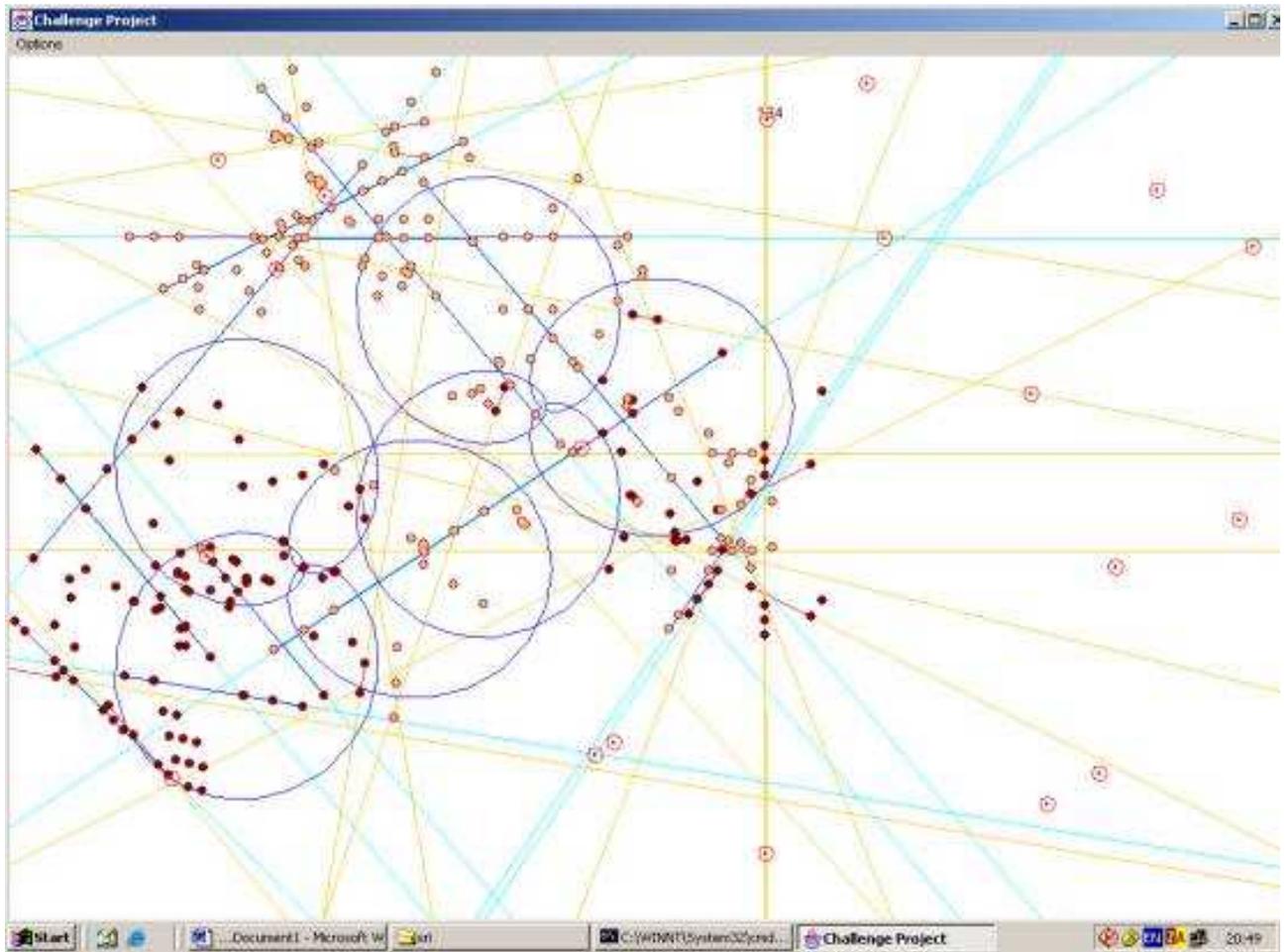


Figure 3.12: Simulating 2 Doppler radars tracking 30 targets. The dots represent sampled target states. The shades of lines represent 100% and 50% tracked targets.

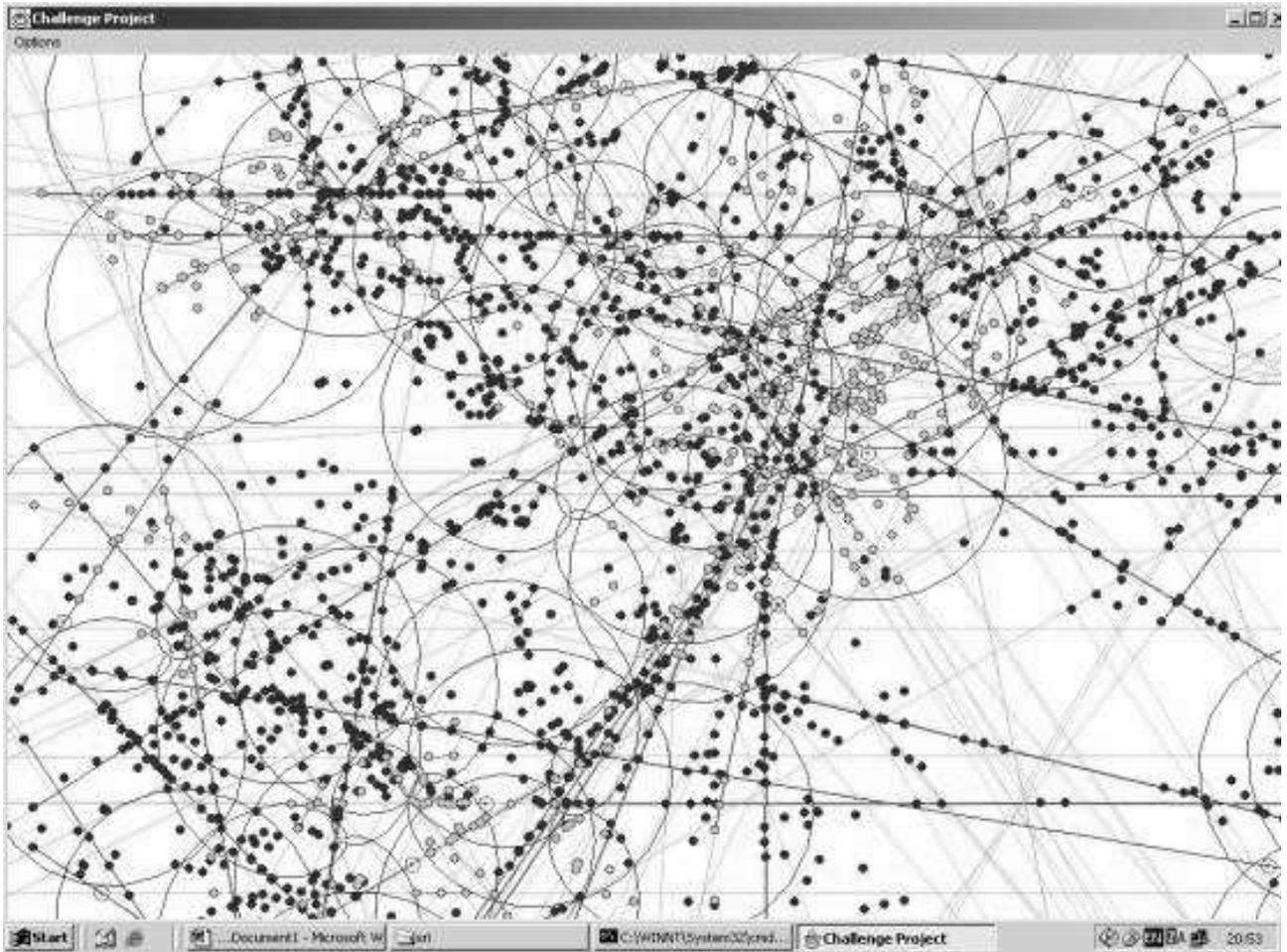


Figure 3.13: Simulating 20 Doppler radars tracking 30 targets. The dots represent sampled target states and the lines represent tracked targets.

In addition, the identified object state functions were divided into two categories. One category involves the association of only a single function with a particular target, chosen to be part of *the infoMap*; those functions were assigned a probability of 100% and corresponded to the actual object state function (we refer to this as the case of *accurate tracking*). The other category involves the association of two possible object state functions with a particular target. In that case, each state function was assigned a 50% probability of corresponding to the actual state function. We refer to those functions as *semi-accurate*. We will say that one set of agents *did better* than another if it reached a higher tracking percentage and a lower tracking time with respect to the accurate functions and the total tracking percentage was at least the same.

The averages reported in the graphs below were computed for one hour of simulated time. The *target tracking percentage time* was calculated by dividing the number of targets that the agents succeeded in tracking, according to the above definitions, by the actual number of targets during the simulated hour. We considered only targets that exited the controlled zone. The *tracking time* was defined as the time that the agents needed to find the object state function of the target from the time the target entered the simulation. Tracking average time was calculated by dividing the sum of tracking time of the tracked targets by the number of tracked targets.

Basic settings: The *basic setting* for the environment corresponded to an area of 10,000 by 10,000 meters. In each experiment, we varied one of the parameters of the environment, keeping the other values of the environment parameters as in the basic settings.

Each Doppler moved one second and stopped for 5 seconds to take 5 measurements. The maximum detection range of a Doppler in the basic setting was 100 meters; the number of Dopplers was 1,000. The controlled area was divided into 1,000 equal rectangles, each 400x250 squared meters. Each patrolling Doppler was assigned to such an area and executed the patrol movement pattern. 1,000 Dopplers with a detection range of 100 meters each, can cover together up to approximately 8,000,000 squared meters, which is 8% of the controlled area.

The number of targets at a given time point was 1,500. In total, during one hour 5,700 targets entered the controlled area and 4,200 of them exited the area.

In the basic setting we used a hierarchy of 4 levels: three levels of zone group leaders and one of sampler group leaders. Each of the zone group leaders divided its zone into 4 areas and assigned a sub-leader to each one of them. Therefore

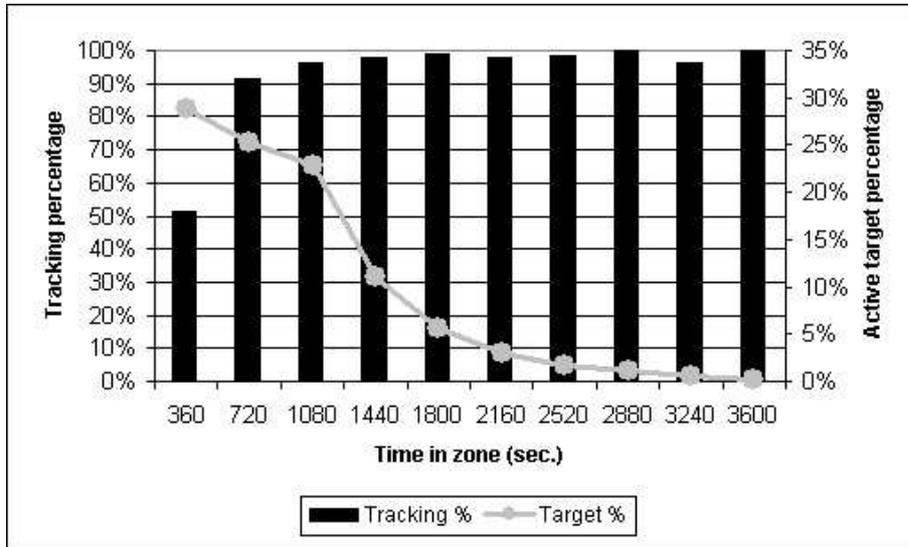


Figure 3.14: Tracking percentage by time in zone (Sec.).

there was one leader at the top level, 4 at the second level, 16 at the third and 64 at the fourth. Each Doppler sensor communicated with one of the fourth-level leaders.

3.4.3 Results

We conducted three sets of tests: (i) evaluating the basic settings, (ii) investigating the influences of the number of levels in the hierarchy, and (iii) studying the tolerance towards faulty sensing agents, leaders and sensing noises. At this state of our research, samplers and leaders do not react to the changes in the functioning agent community.

Basic settings results: Our hypothesis was that by applying the DDM hierarchy model we would be able to very quickly track many targets. We also hypothesized that the tracking period for each target would be significant. We ran the simulation using the basic settings and evaluated the results.

Figure 3.14 shows the percentage of tracked targets as a function of the time each target remained in a zone. To put this histogram in context we added the gray graph that corresponds to the right legend. This graph reflects target distribution with respect to the time spent in the zone.

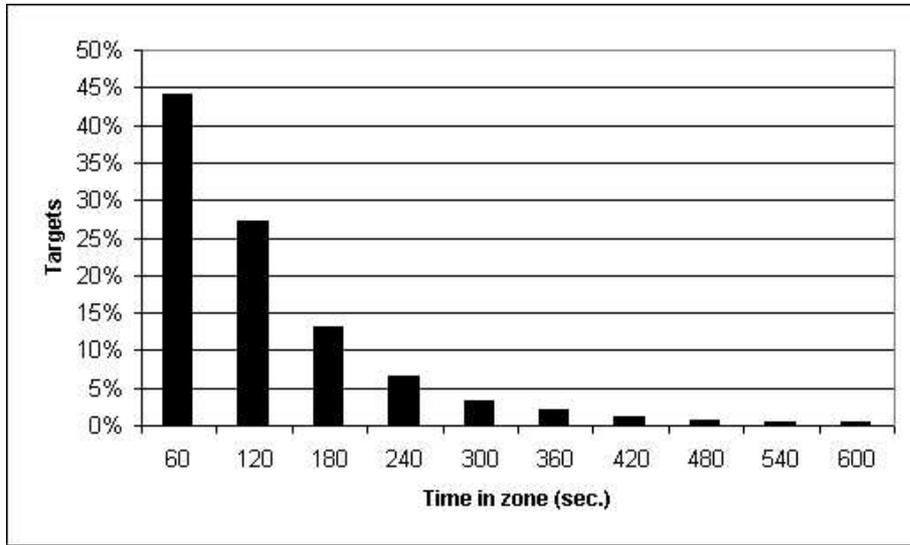


Figure 3.15: Time to track distribution (Sec.).

We can see that the system accurately tracked 83% of the targets. This was achieved with Dopplers covering only 8% of the area. A little more than 50% of the targets that stayed in the controlled zone less than 360 seconds were tracked. Note that most of the targets passing through the simulated area remained in the area less than 720 seconds. During that time the patrol method tracked many targets and therefore achieved rapid tracking.

Figure 3.15 shows the number of targets that were tracked upon entering a zone. Most of the tracking was achieved in less than 2 minutes from the time of a target's entrance into the zone. The system tracked 71% of its tracked targets in this period.

Figure 3.16 plots the tracking duration, which is the period of time between the first and the last time a target was detected. The figure indicates that the system tracks more targets for less duration. However, it tracks most of the tracked targets for more than 6 minutes.

Level comparison: We investigated the influence of the number of levels in the hierarchy. Our hypothesis was that too few levels would overload the leader agents so they would not have enough time to process the information. We also anticipated that, as more leader agents were involved in generating the global solution, a less accurate solution would result.

Figure 3.17 presents the tracking performance of the system as a function of

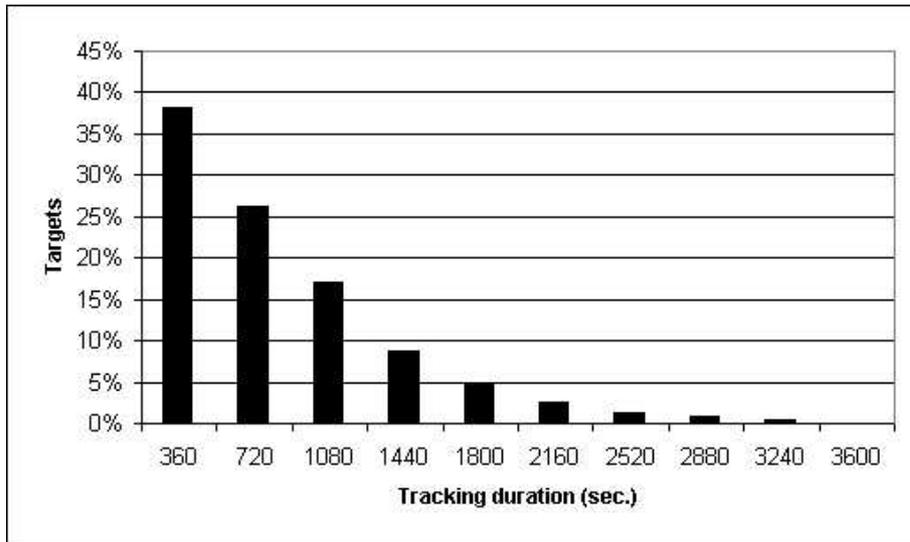


Figure 3.16: Tracking duration distribution (Sec.).

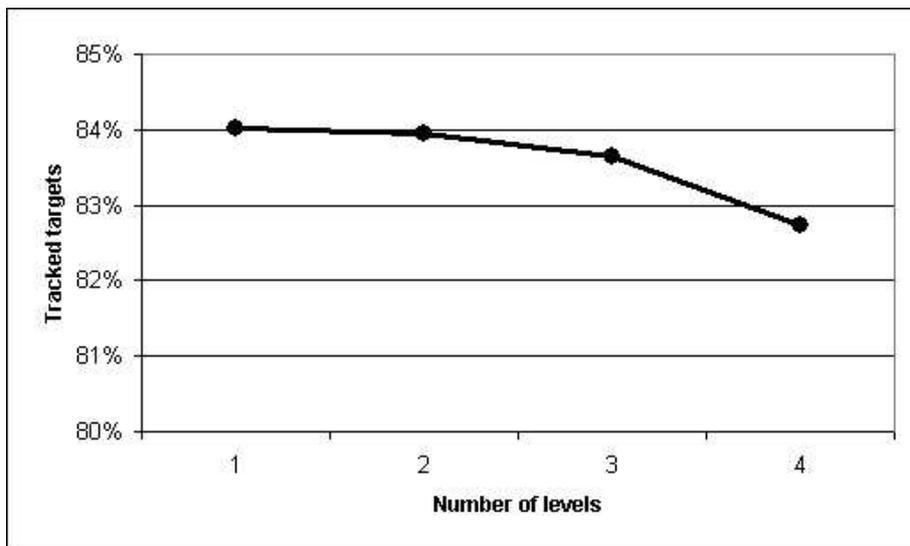


Figure 3.17: Accurate tracked target percentage as a function of the number of levels.

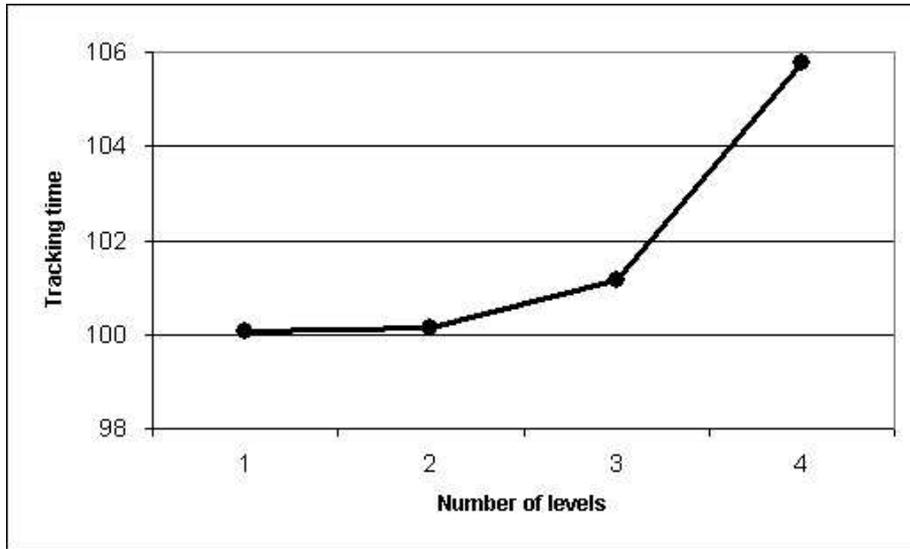


Figure 3.18: Accurate tracking time (Sec.) as a function of the number of levels.

the number of levels in the hierarchy. As we hypothesized, the system tracked less targets as the number of levels increased. This can be explained by a greater fragmentation of the zone, i.e. 4 quarters in 2 levels versus 64 in 4 levels. The figure shows that the decrement is narrow.

As shown in Figure 3.18, the average time to track a target increases as the number of levels increases. However, it increases only from 100 seconds to 106 seconds while the number of levels increases from 1 to 4.

Figure 3.19 presents the duration it took an agent to perform its task. In this figure we present the maximum time when using the computer capabilities as detailed above. The maximum time is very close to the average time; therefore we do not present the latter here. As we predicted, while using only one level the agent will need more time than it has. In our case an agent needed 16,000 seconds (about 5 hours) to process data gathered during 1 hour. That means that in the case of one level the system will not converge. Using 2 levels enabled the system to solve the problem in only 35 minutes. Using 4 levels decreased the maximum time that an agent needed to process data collected in an hour to only 10 minutes.

In Figure 3.20 we show the total number of bytes transferred between agents during one hour, relative to the number of levels in the hierarchy. The capsules generated by samplers and sent to sampler group leaders resulted in a transfer of 4Mb. Having a massive communication load may cause a bottleneck in the

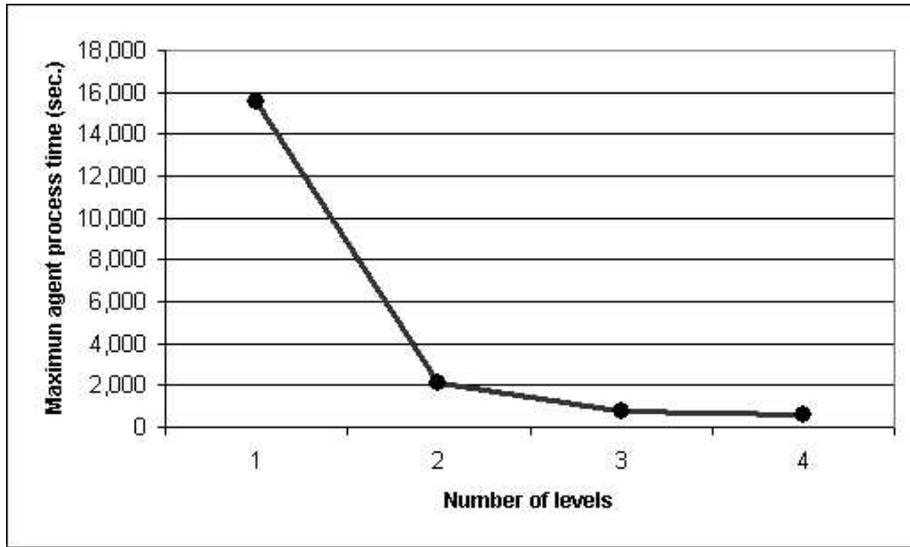


Figure 3.19: Maximum agent process time (Sec.) as a function of the number of levels.

receiving agent that may lead to delays. Moreover, such a bottleneck may result in a loss of important information in case of agents' faults. When adding more levels to the hierarchy, more agents transfer information upwards and therefore the total number of bytes transferred is increased. On the other hand, adding more levels decreases the average number of bytes every agent receives. In figure 3.21 we can see the significance of the reduction of the average communication load on the receiving agent when increasing the number of the levels.

We used a hierarchy formation such that every level has four times more agents than its leader's level. Therefore, the total number of leader agents receiving information in the hierarchy is 1, 5, 21 and 85 when using hierarchy of 1,2,3 and 4 levels. Figure 3.21 show the number of bytes transferred according to the number of agents.

Dysfunctional sampling agents: To investigate the fault tolerance property of the hierarchy model in a large-scale environment we disabled some of the sampling agents. We increased the number of disabled sampling agents from 0% as in the basic settings to 90%, leaving only 10% active agents. We hypothesized that by increasing the number of faulty sampling agents the system would not perform as well as in the basic settings. Our goal was to place a bound on the number of dysfunctional sampling agents that the system could tolerate while still

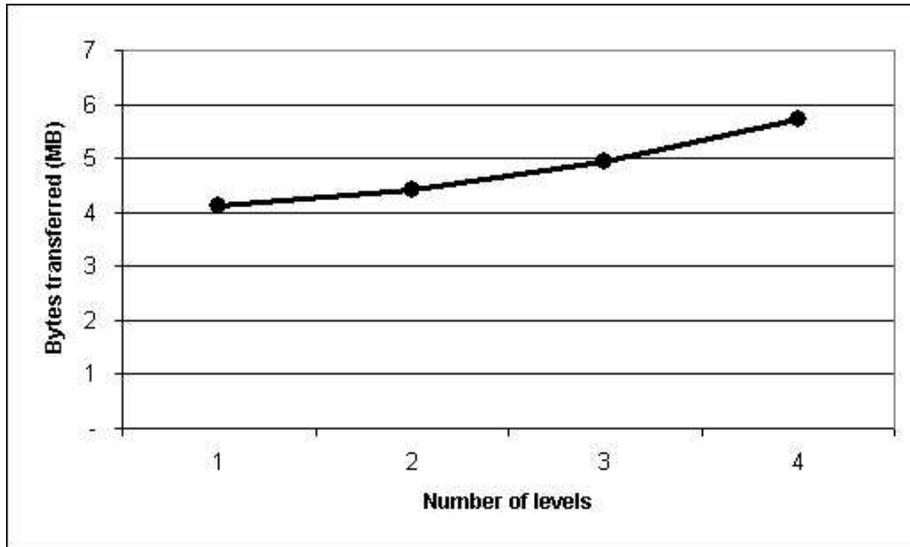


Figure 3.20: Bytes transferred as a function of number of levels.

performing well.

Figure 3.22 shows the accurate tracked targets percentage as a function of the number of samplers which stopped functioning. We found that increasing the number of disabled sampling agents also increases the time it takes to track targets. By increasing the number of disabled sampling agents by 5% the average time it takes to track a target increased by 6%.

Dysfunctional leader agents: A second aspect of the system’s fault tolerance is its response to dysfunctional leaders. In contrast to dysfunctional samplers, a dysfunctional leader will result in a difference in the coverage of the system. For example, consider a case in which a leader responsible for half of the controlled area stops functioning. Using the patrol Doppler movement pattern will result in a loss of information from half of the samplers. We hypothesized that performance would be significantly influenced by this factor. To validate this hypothesis we conducted several simulations in which we varied the number of dysfunctional sampler leaders.

Figure 3.23 confirms our hypothesis. It shows that the system could tolerate a reduction of up to 13% in the number of functioning sampler leaders. A reduction of 18% or more resulted in a very low performance level. However, despite the fact that the system demonstrated poor tracking percentage for high-rate dysfunctional sampler leaders, we discovered that it still tracked targets quickly. We

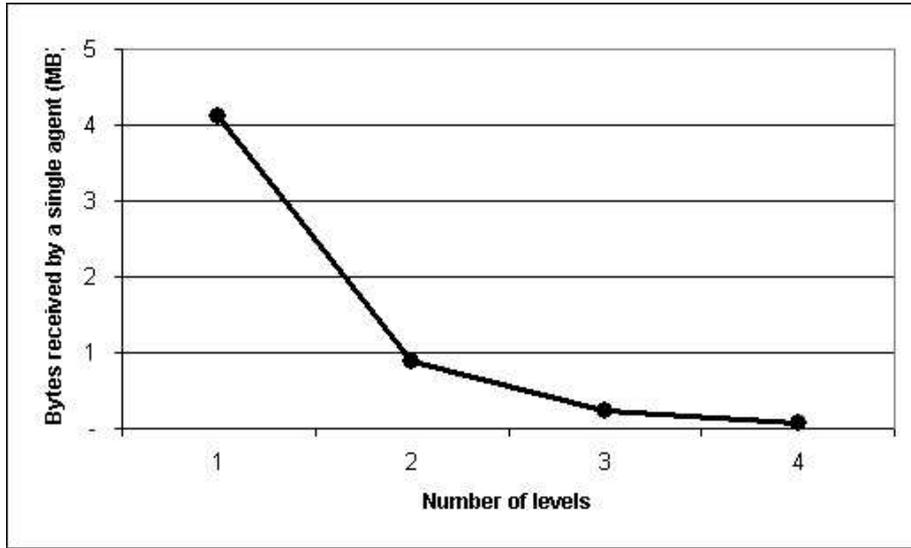


Figure 3.21: Average number of bytes received by a single agent as a function of the number of levels.

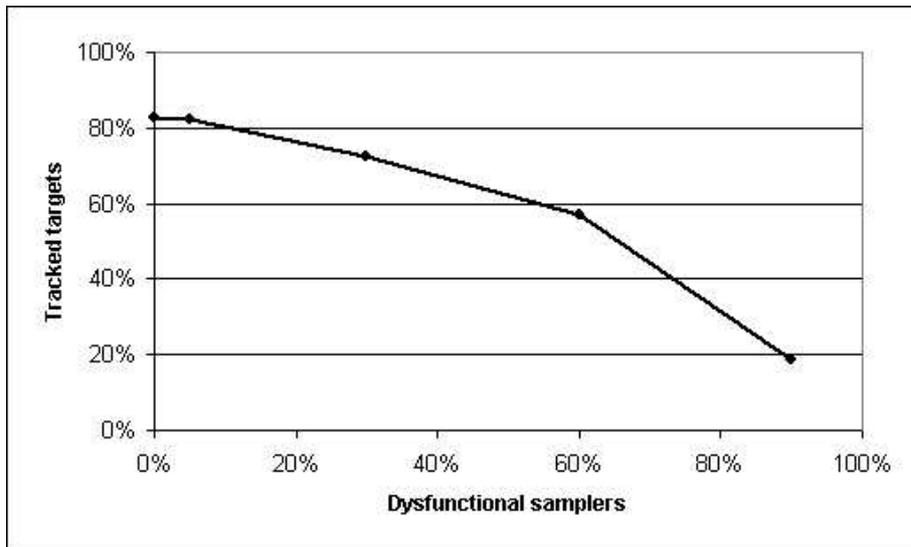


Figure 3.22: Accurate tracked target percentage as a function of dysfunctional samplers.

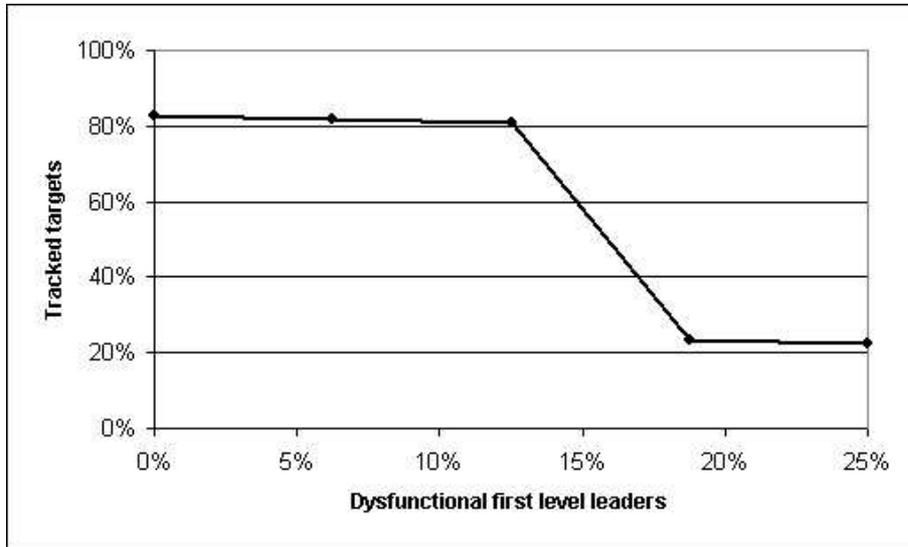


Figure 3.23: Accurate tracked target percentage as a function of dysfunctional first level leaders.

hypothesize that adopting a reactive approach that will enforce division of the area among the active agents will overcome this problem. We plan to report the results of our investigation of this hypothesis in a future document.

Noisy communication: As we stated, we would like to show that using simple and cheap sensors may be beneficial even if they tend to malfunction or if communication with their leaders degrades. We conducted a thoughtful simulation testing the system while using faulty communication between samplers and leaders. We predicted that the system would be tolerant towards such noise.

We found, as shown in Figure 3.24, that even if 50% of the messages did not reach their destination (either because of faulty communication or faulty samplers), the system still performed well. Losing 50% of the messages resulted in a reduction of only 5% of the tracked targets and increased the tracking time by 20 seconds.

3.4.4 Small Scale Results

We also tested the DDM in a small-scale environment to compare various properties of the suggested solution. We tested the need for hierarchical structure and sensor mobility.

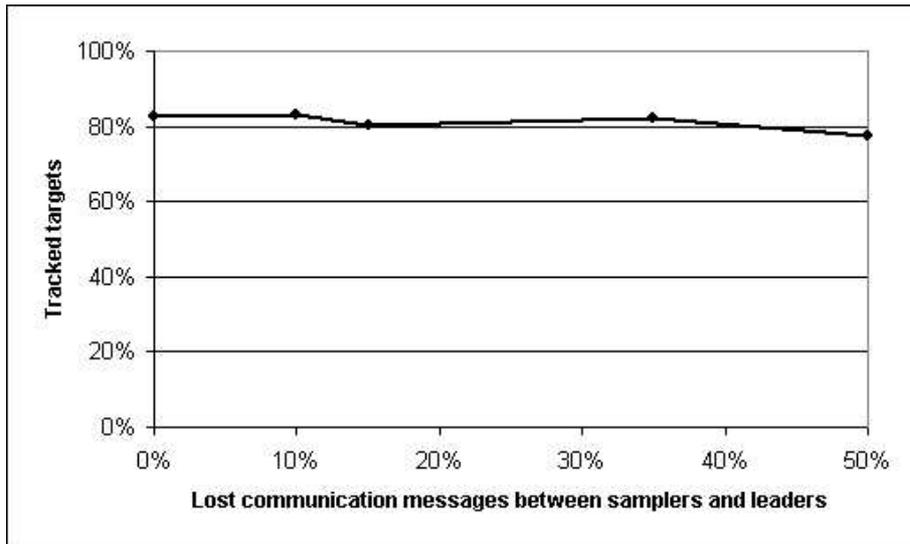


Figure 3.24: Accurate tracked target percentage of patrol as a function of lost communication messages between samplers and leaders.

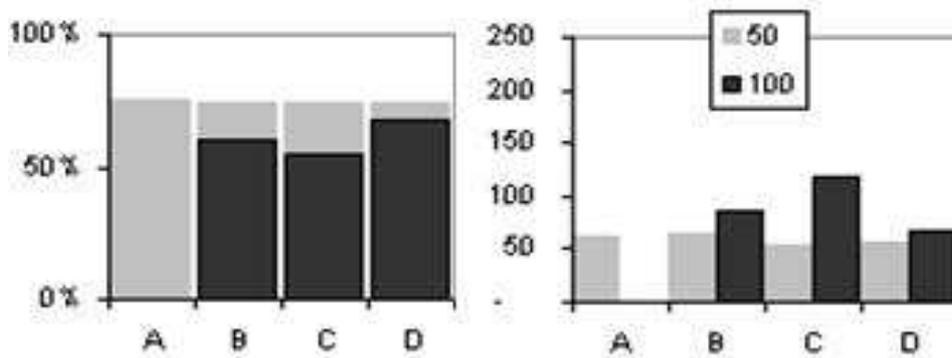


Figure 3.25: Target tracking percentage and average time by the settings.

Basic Settings: The basic setting of the environment was 1200 by 900 meters. In the experiments, we varied one of the parameters of the environment, keeping the other values of the environment parameters as in the basic settings. The Dopplers were mobile and moved randomly as before. Each Doppler stopped every 10 seconds, varied its active sensor randomly, and took 10 measurements. The maximum detection range of a Doppler in the basic setting was 200 meters; the number of Dopplers was 20 and the number of targets at any given time was 30. The DDM hierarchy consisted of only one level. That is, there was one sampler-leader that was responsible for the entire area.

We first varied several settings of both the hierarchy model and the sampling agents. Each setting corresponded to (1) whether a hierarchy model (H) or a flat model (F) was employed; (2) whether the sampler-agents were mobile (M) or static (S); and (3) whether Dopplers varied their active sectors from time to time (V) or used a constant one all the time (C). In the flat model the sampler agents used their local capsules to produce task state functions locally. We refer to a particular combination of settings by combining the labels corresponding to each setting; for example, the label “FSC” would correspond to a flat, static model in which Dopplers kept their active sensor constant.

Mobile and dynamic vs. static Dopplers: In preliminary simulations we experimented with all combinations of parameters (1)-(3) above. In each setting, keeping the other two variables fixed and varying only the mobility variable, the mobile agents did better than the static ones (with respect to the evaluation definition above).

Hierarchy vs. flat models: We examined the characteristics of 4 different settings: (1) FSC that involves static Dopplers with a constant active sector using a non-hierarchical model; (2) HSC as in (A) but using the hierarchical model; (3) FMV with mobile Dopplers that vary their active sectors from time to time, but with no hierarchy; (4) H MV as in (3) but using the hierarchical model. We tested FSC in two experimental settings: one in which Dopplers were located randomly and one in which Dopplers were arranged in a grid formation to achieve better coverage. There was no significant difference between these two FSC formations. Our hypothesis was that the agents in H MV would do better than in all the other settings.

The first finding is presented in the left part of Figure 3.25 This finding indicates that the setting does not affect the overall tracking percentage (i.e., the

tracking percentage of the 50% and 100% functions). The difference between the settings is with respect to the division of the detected target between accurate tracking and mediocre tracking. HMV performed significantly better than the other settings. It found significantly more 100% functions and did it faster than the others. This supports the hypothesis that a hierarchical organization leads to better performance. Further support for a hierarchical organization comes from HSC being significantly better than FMV even though, according to our preliminary results, HSC uses Dopplers that are more primitive than the Dopplers FMV.

Another aspect of the performance of the models is the average tracking time as shown in the right part of Figure 3.25. Once again, one can see that the hierarchically based settings lead to better results. We found that by considering only targets that stayed in the controlled zone at least 60 seconds, HMV reached 87% tracking percentage and 83% were accurately detected.

We also studied the performance of hierarchies with two levels: one zone leader leading four sampling leaders. The area was divided equally between the four sampling leaders, and each obtains information from the many mobile sampling agents located in its area. In that configuration Dopplers were able to move from one zone to another; Dopplers changed their sampling leader every time they moved from one zone to another. Comparing the results of the two level hierarchy simulations, with the one level hierarchy simulations we found that there was no significant difference in performance (with respect to the evaluation definition) of a system composed of two levels with a system consisting of only a one level hierarchy. However, consistent with theorem 1, the computation time of the system was much lower.

Communication and noise: While the performance of the hierarchy-based models are significantly better than the non-hierarchy ones, the agents in the hierarchy model must communicate with one another, while no communication is needed for the flat models. Thus, if no communication is possible, FMV should be used. However, with communication, messages may be lost or corrupted. Recall that the data structure exchanged in messages is the capsule; in our simulations using a hierarchy model, each sampling agent transmitted 168 bytes per minute. We studied the influence of randomly corrupted capsules on the HMV's behavior. Figure 3.26 shows that as the percentage of the lost capsules increases the number of tracked targets decreases; however, up to a level of 10% noise, the detection percentages decreased only from 74% to 65% and the accurate tracking time increased from 69 seconds to only 80 seconds. Noise of 5% results in a smaller decrease to a tracking

accuracy of 70% while the tracking time is slightly increased to 71. DDM could even manage with noise of 30% and track 39% of targets with average tracking time of 115 seconds. In the rest of experiments we used the HMV settings without noise.

Varying the number of Dopplers and targets: We examined the effect of the number of Dopplers on performance. We found that, when the number of targets is kept fixed, as the number of Dopplers increases the percentage of accurate tracking increases. This result demonstrates that the system can make good use of additional resources that it might be given. We also found out that as the number of Doppler sensors increases, the 50% probability paths decrease. That may be explained by the fact that 100% paths result from taking into consideration more than one point of view of samples. We also found that increasing the number of targets, while keeping the number of Dopplers fixed does not influence the system's performance. We speculate that this is because an active sector could distinguish more than one target in that sector.

Maximum detection range comparison: We also tested the influence of the detecting sector area on performance. The basic setting uses Dopplers with a detection range of 200 meters. We compared the basic setting to similar ones with detection ranges of 50, 100 and 150 meters. We found that as the maximum range increases the tracking percentage increases up to the range of covering the entire global area. As the maximum radius of detection increases the tracking average time decreases. This is a beneficial property, since it suggests that better equipment (i.e., sensors and other hardware) will lead to better performance.

3.5 Related work

The benefits of hierarchical organizations have been argued by many. So and Durfee draw on contingency theory to examine the benefits of a variety of hierarchical organizations; they describe a hierarchically organized network monitoring system for task decomposition and they also consider organizational self-design [[So 1996]]. DDM differs in its organization use to dynamically balance computational load and also in its algorithms for support of mobile agents.

The idea of combining partial local solutions into a more complete global solution goes back to early work on the Distributed Vehicle Monitoring Testbeds (DVMT) [[Lesser 1987]]. DVMT also operated in a domain of distributed sensors

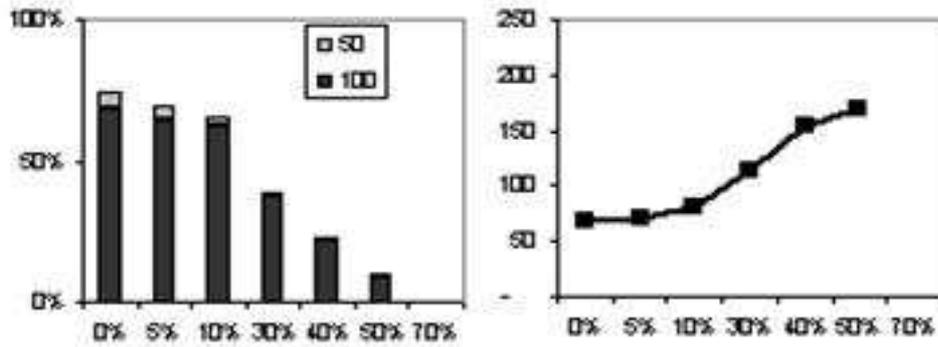


Figure 3.26: Target detection percentage and average time as function of the communication noise.

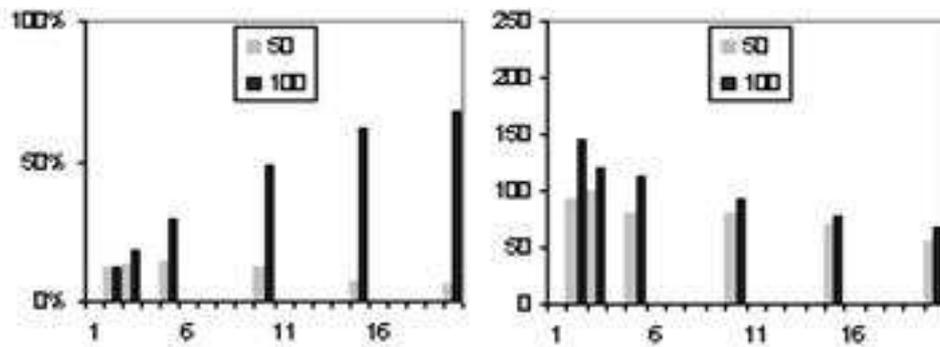


Figure 3.27: Tracking percentage and average time as a function of the number of Dopplers.

that tracked objects. However, the algorithms for support of mobile sensors and for the actual specifics of the Doppler sensors themselves is novel to the DDM system. Within the DVMT, Corkill and Lesser investigated various team organizations in terms of *interest areas* [[Corkill 1983]] which partitioned problem solving nodes according to roles and communication, but they were not initially hierarchically organized [[Ishida 1992, Scott 1992]]. Wagner and Lesser examined the role that knowledge of organizational structure can play in control decisions [[Wagner 2000]].

All of the other approaches discussed in this volume assume that agents are stationary. Those approaches make use of measurements from three Doppler sensors, taken at the same time, and intersect the arcs corresponding to each sensor. The intersection method depends on the coordinated action of three Doppler sensors to simultaneously sample the target. Such coordination requires good synchronization of the clocks of the sensors and therefore communication among the Doppler agents to gain that synchronization. In addition, communication is required for scheduling agent measurements. We have described an alternative that can make use of uncertain measurements; we focus on the combination of partial and local information. Note, that even though our agents associate a time stamp with each capsules, DDM does not require that the sensors are fully time synchronized. The ResBy relation may allow small deviation of the time. For example: $ResBy(\langle t1, s1 \rangle, \langle t2, s2 \rangle)$ may be $s1.v == s2.v \& (s1.x - t1 * s1.v) - (s2.x - t2 * s2.v) \leq \epsilon$ Using a large ϵ may indicate high tolerance towards non synchronized clocks. However, increasing the value of epsilon increases the probability to identify two different targets as the same one.

In their work, Yu and Cysneiros [[Yu 2002]] describe challenges related to large-scale information systems. They claim that large-scale systems have the potential to support greater diversity, offering more flexibility and better robustness as well as more powerful functionalities compared to traditional software technologies. In our work we address these challenges and propose an efficient solution.

Silva *et al*, have developed the Reflective Blackboard architectural pattern for large-scale systems [[Silva 2002]]. This is the result of the composition of two other well-known architectural patterns: the Blackboard pattern and the Reflection pattern. They separate control strategies from the logic and data. In our work we use independent agents that act autonomously. Such a loose coupling is beneficial in terms of simplicity, robustness and fault tolerance.

Tel has studied the performance of a network tree with n processors providing communication between every pair of processors with a minimal number of links

(n-1) [[Tel 1991]]. The communication complexity in a tree topology is influenced by the diameter of the number of levels in the tree. Therefore a tree with fewer levels will have a better communication complexity. However each node has more computations to perform and can therefore become a bottleneck. A failure of a node will split the tree into a larger number of unconnected subsets. In the work we have described, we have investigated the relation between the number of levels in a hierarchical structure and performance; we have presented suggestions of how to choose the right number of levels.

3.6 Conclusions

We have shown that problems involving hundreds and thousands of Dopplers and targets cannot be solved using a traditional flat architecture. Our approach was based on distributing the solution into smaller problems that could be solved partially by simple agents; agents are, in addition, organized hierarchically. Using many simple and cheap agents instead of a much smaller number of sophisticated and expensive ones may also be cost-effective: it is often more affordable to replace and maintain many simple agents than to depend on a few sophisticated ones. Our approach incorporated methods for combining partial solutions to form a global solution, and involved an autonomous movement algorithm that could be executed by each sampling agent.

The number of levels in a hierarchy was shown to influence accuracy. As the number of levels increased the number of tracked targets dropped, even though this drop was moderate. However, as the number of levels increased, the time needed by every agent to complete its mission dropped exponentially. By combining these two results we were able to balance both properties. Choosing the right number of levels should also take into consideration the time it takes to track targets. As we have shown, it takes more time to track targets as the number of levels in the hierarchy is increased.

To conclude, we have shown that a large-scale ANTs system can perform well even if agents are very simple and inaccurate. We have shown how partial information can be combined and how the existence of dysfunctional participants can be overcome.

Chapter 4

Multi-channel communications scheduling

The ANTs challenge problem relies on wireless communication between a set of sensors. Each sensor is equipped with a low bandwidth transceiver and eight distinct radio channels are available for communication. Each sensor can be programmed to send and receive on any of the eight channels; the transmission and reception channels of each can be different. A sensor cannot send and receive at the same time. Efficient use of the bandwidth requires good synchronization between sending and receiving nodes to ensure that they are tuned to the right channel at the right time.

SRI researched scheduling algorithms based on graph coloring for scheduling communication between sensors. This approach assumes that sensors have a limited communication range so that transmissions from two sufficiently distant sensors do not interfere, even if they use the same band. Unfortunately, this assumption is not valid for the hardware developed for the ANTS challenge problem.

In a second phase of the project, we investigated a different approach to sharing the RF medium, by developing a channel reservation protocol for ANTs-like networks. We experimented with a prototype implementation using the challenge problem simulator. Unfortunately, performance was poor because of the impossibility of asynchronously receiving and reacting to RF messages within a Java virtual machine, and the limitations of the Java scheduling model.

4.1 Scheduling Access to Radio Channels

4.1.1 Model

A network of sensors that communicate via wireless links can be modeled naturally as a graph $G = (V, E)$:

- Each vertex v of V represents one of the sensors.
- There is an edge between two vertices v and v' if the two sensors v and v' are within communication range of each other.

The set of neighbors of a node v , that is, the set of nodes within communication range of v is denoted by $E(v)$.

This model assumes that communication links are symmetric: if v can send to v' then v' can send to v . In the common case, the sensors are placed on a two dimensional plane, and two sensors are within communication range of each other if the distance between them is less than a bound r . This gives rise to a special class of graphs called *point graphs* in [Sen and Luson 1997] or *disk graphs*.

We assume that a schedule is composed of a sequence of time slots. In each time slot, a node is either idle, sending, or receiving a message on a specified channel. The schedule must ensure that sender and recipients of a message are synchronized and prevent message collisions. Such a schedule can be effective only if the communication pattern between the sensors is sufficiently regular and predictable. We assume that the communication pattern is periodic and can be described as follows. During each period, a finite set M of messages must be transmitted. Each message m has a unique sender $s(m) \in V$ and a set of recipients $r(m) \subset V$. We assume that the nodes of $r(m)$ are all neighbors of $s(m)$, that is, $r(m) \subseteq E(s(m))$. In other words, we ignore multihop routing issues.

In this model, a communication schedule is defined by the number n of time slots it contains and by the time slot $t(m)$ and the band $b(m)$ assigned to each message $m \in M$. The time slot $t(m)$ is an integer between 1 and n , and $b(m)$ is one channel. In time slot $t(m)$, node $s(m)$ must be sending m on band $b(m)$ and all nodes in $r(m)$ must be listening on band $b(m)$ to receive the message. For these mappings to define a valid schedule, the two following constraints must be satisfied:

1. If $m_1 \neq m_2$ and $t(m_1) = t(m_2)$ then the two sets $A(m_1) = r(m_1) \cup \{s(m_1)\}$ and $A(m_2) = r(m_2) \cup \{s(m_2)\}$ are disjoint.

2. If $m_1 \neq m_2$, $t(m_1) = t(m_2)$, and $b(m_1) = b(m_2)$ then $r(m_1) \cap E(s(m_2)) = \emptyset$.

The first constraint guarantees that two messages sent in the same slot have no recipient in common, that their senders are distinct, and that the sender of one is not an intended recipient of the other. The second constraint prevents collision between two messages. If m_1 and m_2 are sent on the same band in the same slot, then the recipients of m_1 must not be within communication range of $s(m_2)$, the sender of m_2 .

If the objective is to maximize throughput, one must construct two mappings, b and s , that satisfy the above constraints, while minimizing the number of slots n . Alternatively, a bound T (i.e., the communication period) may be specified a priori, and one attempts to construct a schedule with $n \leq T$. Both problems can be solved using general CSP algorithms.

4.1.2 Special Case: Broadcast

An interesting special case is when we have $r(m) = E(s(m))$ for all the messages m . In this case, the recipients of any message from v are all the neighbors of v ; any message from v is broadcast to all nodes within communication range of v . This mode of communication, based on broadcast, is very efficient as it maximizes the number of nodes that a message reaches while minimizing the number of transmissions. In this broadcast mode, it is easy to see that constraint (1) above implies constraint (2), which means that the channel assigned to any message m is irrelevant. Thus, if all messages from every node are broadcast to all its neighbors, there is no advantage in using more than one channel. Two nodes that transmit in the same time slot must have no neighbor in common and can then use the same frequency band.

Given a graph $G = (V, E)$ representing a sensor network, let $G^2 = (V, E')$ be the square of G . There is an edge between two vertices v and v' of V in G^2 if and only if the distance between v and v' in G is at most two. In other words, there is an edge between v and v' in G^2 if v and v' are adjacent or have a common neighbor in G . Since the band assignment b is irrelevant, constructing a schedule amounts to assigning a time slot to each message so that constraint (1) is satisfied: if $m_1 \neq m_2$ and $t(m_1) = t(m_2)$ then $A(m_1) \cap A(m_2) = \emptyset$. By definition of G^2 , we have $A(m_1) \cap A(m_2) \neq \emptyset$ if and only if $s(m_1)$ and $s(m_2)$ are adjacent in G^2 . If the communication pattern M contains exactly one message from each node, we can identify the message m and the node $s(m)$. In this case, the mapping t

is nothing other than a (proper) vertex coloring of G^2 . Constructing a schedule amounts then to finding a minimal coloring of G^2 . Even if the message pattern M contains distinct messages with the same sender, one can transform the graph G and the pattern M so that a coloring of G^2 gives a communication schedule.

Coloring the square of a graph, and several generalizations, has been extensively studied in a context related to ours, namely the assignment of non-interfering radio channels to base stations of a cellular network. If G has maximal degree Δ , then obviously a coloring of G^2 requires at least $\Delta + 1$ colors. Conversely, it is easy to see that Δ^2 colors are sufficient.

Given a graph G , finding a minimal coloring for G^2 is NP-hard for general graphs [McCormick 1983]. The problem becomes polynomial for restricted classes of graphs (for example, when G is a tree or is obtained by assuming all the nodes are located on a straight line) [Sen and Luson 1997]. On the other hand, the problem is known to be NP-hard for point graphs [Sen and Luson 1997], the class of graphs that model ANTs-like sensor networks. It is also NP-hard for planar graphs [Ramanathan and Lloyd 1992].

4.1.3 Algorithms and Experiments

We have implemented and experimented with two graph-coloring algorithms for constructing communication schedules. These algorithms take as input a graph G , compute its square, and attempt to construct a minimal coloring of G^2 . Our first prototype uses the Chaff SAT solver [Moskewicz et al 2001]. It encodes the coloring problem into a boolean satisfiability problem that is then solved by Chaff. A main limitation of this approach is that the boolean solver does not take into account the many symmetries present in the problem. A better branch and bound coloring algorithm was then implemented. It uses a standard heuristic (DSATUR) and starts from an initial partial coloring obtained from a maximal clique of G^2 .

Example results from these algorithms are given in Table 4.1. These results were obtained for a fixed set of 100 sensors placed 10 units apart on a 10×10 square grid. Different graphs G were then obtained by varying the communication range of the sensors. Each row of the table shows the maximal degree of G (that is, the largest number of sensors within communication range of a single node), and the minimal number of colors found and the search time for both algorithm. The branch and bound algorithm always finds the minimal coloring on these examples, but the SAT solver fails (after several hours of search) on the last example, where the communication range is 31 units.

These examples and further experiments show that straightforward coloring

Range	Max Degree	Coloring	SAT alg.	Branch/Bound
12	4	5	0	0
15	8	9	0	0
21	12	13	0	0
23	20	23	11.18s	8.17s
29	24	25	0	0
31	28	33	-	81.42s

Table 4.1: Experiments on a 10×10 Sensor Grid

algorithms can find optimal communication schedules for sensor networks of up to a few hundred nodes, provided the communication range (and thus the maximal degree of the graph G) is relatively small. The branch and bound algorithm is much more efficient than SAT encoding when the communication range increases.

Although these initial results showed the potential and feasibility of constructing communication schedules for multichannel wireless sensor networks, we did not pursue this direction further because of a mismatch between our assumptions and the ANTs challenge-problem hardware. The sensors developed for ANTs have a communication range that is very large compared with their sensing range. As a result, in any practical configuration, all the sensors are within communication range of each other. The corresponding graph G is then a complete graph, and communication scheduling in such a case reduces to a trivial round robin. A second limitation is that scheduling requires an a priori knowledge of the communication patterns in the network. Unfortunately, it is very difficult to predict or bound how much communication will occur when sensors cooperate and negotiate using complex algorithms.

4.2 A Channel Reservation Protocol

As an alternative to channel scheduling, we investigated an access control protocol based on channel reservation. This protocol uses channel 0 as a control channel, while the other channels are used for communicating application data between sensors. The protocol allows a sensor to reserve one channel for communication between itself and a set of other sensors. The intent of this protocol is primarily for an ANTs tracker agent to select a channel on which it will receive data from a set of sensing agents for tracking purposes.

Using this protocol, each node in the network maintains a table T that keeps track of the current reception channel of all the other nodes. A network node can be in one of two states: in the default *idle* state, a node is listening on the control channel, namely channel 0. In the *active* state, a node has been allocated a specific reception channel (other than 0) and is listening on this channel. Initially, all nodes are idle.

When a node v becomes active, it first selects a free channel (if any), then requests to be allocated that channel by broadcasting a control message on channel 0. If no message collision occurs, all currently idle nodes receive the request and one of them sends an acknowledgment on the control channel. The reception of this acknowledgment indicates to v that its request is granted: v moves to the active state and switches to its new listening channels. Since both the request and the acknowledgment are broadcast on channel 0, all the idle nodes receive them and can determine that v has changed channel and update their table T accordingly. The protocol attempts to maintain the tables T of all the idle nodes consistent and up to date. All idle nodes can then determine locally the channel to use when they need to send data to v . The common table T is also used to determine which node should respond to requests on the control channel. A specific node x_T is selected from T and is in charge of responding to all requests sent on channel 0. A second node y_T is in charge of responding to requests that originate from x_T itself. Both x_T and y_T are selected using an arbitrary rule among the idle nodes of T (e.g., the two idle nodes of smallest indices). An active node cannot keep its copy of T up to date since it does not receive the control messages and acknowledgments. If an active node needs to send data to v , it first requests an up-to-date copy of T on channel 0, this fresh copy is broadcast by x_T .

Message collisions may occur on channel 0 if two nodes send requests simultaneously. However the protocol attempts to minimize the probability of collisions by limiting control traffic. Control messages (requests and responses from x_T or y_T) are small to reduce the chance of collisions. In addition, all idle nodes listen to the control traffic and can determine whether or not channel 0 is busy. If an idle node observe that a request has been sent by another node and has not been acknowledged, it will delay its own requests until the expected acknowledgment is transmitted. These mechanisms reduce but do not eliminate message collisions on the control channel. However, a node that has sent a request can determine via a timeout mechanism that a request has not been received (when no acknowledgment is received) and thus whether it has succeeded. A random backoff and retry mechanism allows a node to resend a request in such a case.

We have experimented with a prototype implementation of this protocol in

Java for the challenge problem simulator (radsim). Unfortunately, performance was disappointing. The main difficulty was that Java and the radsim simulator do not give any means to quickly react and respond to messages received on the control channel. The Java implementation is multithreaded. An RF message is first obtained by a Java listening thread that polls the hardware driver, it is then copied and buffered for another Java thread to process it. Both threads compete with each other, with other Java threads inside the JVM, and with non-Java processes running on the same machine. The delay between a message being received by the hardware and its processing by the software depends on how and when the listening thread is active. The JVM and operating system do not provide ways to accurately schedule the listening thread. As a result, we never managed to reliably make the responder node T_x receive all requests on channel 0 and send acknowledgments in a timely manner. Delayed acknowledgments are disastrous as they cause many requests to fail and can lead to distinct nodes having inconsistent views of the table T . Furthermore, it is difficult to ensure that all idle nodes have all processed the same messages as scheduling delays vary across machines, which again causes inconsistencies.

4.3 Paper on Dynamic Scan Scheduling

A paper describing the results of our previous research in the ANTs program was written and presented at the IEEE Real-Time Systems Symposium, at Austin Texas, in December 2002.

Chapter 5

Summary and conclusions

The table in Figure 5.1 summarizes our contributions to the solution of the problems posed in Chapter 1. We repeat here the desiderata set forth in Chapter 1.

Distributed: Any resource allocation algorithm should be distributed in the sense that it should not depend on some centralized repository of global information where allocation decisions must be made. Such an assumption would be overly restrictive given the constraints imposed within real world settings in which inter-agent communications may be limited.

Incremental and realtime: The time-stressed nature of realworld problem domains precludes the possibility of computing optimal resource allocations before execution. Instead, agents should negotiate partial, good-enough allocations which can later be refined if time permits.

Flexible task allocation: Task allocation mechanisms should be flexible in the sense that potential allocations can be explored either sequentially, in terms of possible task-resource pairs, or combinatorially in the form of sets of multiple tasks and resources. In the latter case, mechanisms should be able to deal with tasks that can interact: that is, in which the cost of doing several tasks is not simply the sum of the individual costs of each task.

Adaptive resource allocation: Dynamic problem settings in which new tasks or resources can appear (or disappear) during the allocation process require that it not be necessary for allocation processes to be re-started from scratch each time the global situation changes.

Adaptive Communications: Since bandwidth communications are assumed to vary, allocation algorithms should be able to adapt to limits imposed by the communications medium.

Fault tolerant: Any solution should be fault tolerant in the sense that it should be adaptable to resource loss during execution, as opposed to requiring that the allocation be re-started from scratch.

Scalable: Any solution should be scalable to very large agent and task settings.

The left-most column of Figure 5.1 refers to the above requirements. The columns summarize the various algorithms and approaches we have explored. As shown, the best results were obtained with either the dynamic mediation algorithm and a combination of a team commitment architecture and monitoring process; or the DDM system with hierarchical team structure. In the case of mediation, we did not run any experiments on large scale systems. DDM is less flexible than mediation since the “negotiation” is kept to an absolute minimum.

	auction	combi- natorial auction	combi- natorial alloca- tion	team commit- ments	dynamic media- tion	monitor- ing	organi- zational structure
Distributed	X	X	X		X		X
Incremental and real- time	X		X		X	X	X
Adaptive resource allocation				X	X		X
Flexible task allocation		X	X		X		
Adaptive communi- cation	X	X	X	X	X	X	X
Fault tol- erant				X	X	X	X
Scalable	X				?		X

Figure 5.1: Meeting the desiderata: note that this table highlights the major elements (left-hand column) addressed by each approach (top row) examined in this report. The column “combinatorial allocation” refers to the algorithm described in Chapter 2.

Bibliography

- [Corkill 1983] D. Corkill and V. Lesser “The use of meta-level control for coordination in a distributed problem solving network,” in proceedings of the International Joint Conference on Artificial Intelligence, 1983, pages 748–756.
- [Dechter 1988] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proc. of AAAI*, pages 37–42, 1988.
- [Dolev 2000] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.
- [Doyle *et al* 1991] Jon Doyle, Yoav Shoham, and Michael Wellman. A logic of relative desire. In *Symposium Methodologies for Intelligent Systems*, pages 16–31, 1991.
- [Durfee 1999] Edmund H. Durfee. *Distributed problem solving and planning*, pages 121–164. MIT Press, 1999.
- [Ephrati and Rosenschein 1993] E. Ephrati and J. Rosenschein. Multi-agent planning as a dynamic search for social consensus. In *IJCAI*, pages 423–429, 1993.
- [Feynman 1963] R.P. Feynman. *The Feynman Lectures on Physics*. Addison-Wesley Publishing Company, chapters 12-14, Bombay, India, 1963.
- [Fischer *et al* 1995] Klaus Fischer, Jörg P. Muller, Markus Pischel, and Darius Schier. A model for cooperative transportation scheduling. In *ICMAS*, pages 109–116, 1995.
- [Hunsberger and Grosz 2000] Luke Hunsberger and Barbara J. Grosz. A combinatorial auction for collaborative planning. In *ICMAS*, 2000.

- [Ishida 1992] T. Ishida, L. Gasser, and M. Yokoo, “Organization self design of production systems,” in *IEEE Transactions on Knowledge and Data Engineering*, 4(2): 123-134, 1992.
- [Lesser 1987] V. R. Lesser, D. D. Corkill and E. H. Durfee, “An update on the Distributed Vehicle Monitoring Testbed”, Computer Science Technical Report, University of Massachusetts, Amherst, 1987, 87-111.
- [Lesser et al 2003] Lesser, V. and Ortiz, C. and Tambe, M. *Distributed Sensor Networks: a multiagent perspective* Kluwer Publishing 2003.
- [McCormick 1983] McCormick, S. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Mathematical Programming*, pages 26:153–171, 1983.
- [Minton *et al* 1992] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for csp. *Artificial Intelligence*, pages 161–205, 1992.
- [Moskewicz et al 2001] Moskewicz, M. and Madigan, C. and Zhao, Y. and Zhang, L. and Malik, S. Chaff: Engineering and Efficient SAT Solver. In *Proceedings of the 39th Design Automation Conference* 2001.
- [Ortiz 1999] Charles L. Ortiz. A commonsense language for reasoning about causation and rational action. *Artificial Intelligence*, 111:73–130, 1999.
- [Ortiz 2001] C. L. Ortiz, E. Hsu, M. desJardins, T. Rauenbusch, B. Grosz, O. Yadgar, and S. Kraus. “Incremental negotiation and coalition formation for resource-bounded agents,” in Proceedings of the AAAI Fall Symposium, 2001.
- [Ortiz and Hsu 2002] Charles L. Ortiz and Eric Hsu. Structured negotiation. In *ICMAS 2002*, 2002.
- [Parkes 1001] David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, May 2001.
- [Poss 1992] Pöss, Christian Doppler in Banska Stiavnica, in *The Phenomenon of Doppler*, Prague, 1992.

- [Ramanathan and Lloyd 1992] Ramanathan, S. and Lloyd, E. On the complexity of distance-2 coloring. In *Proceedings of the 4th International Conference on Computing and Information* pages 71–74, 1992.
- [Rauenbusch 2003] Timothy Rauenbusch. A decision making procedure for collaborative planning. In *AAAI 2003*, 2003.
- [Sandholm 1998] Thomas W. Sandholm. Contract types for satisficing task allocation. In *AAAI Spring Symposium*, 1998.
- [Sandholm 1999a] Thomas W. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.
- [Sandholm 1999b] Tuomas W. Sandholm. Distributed rational decision making. In *Multiagent Systems*, pages 201–258. MIT Press, 1999.
- [Sandholm and Suri 2000] T. Sandholm and S. Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *National Conference on Artificial Intelligence (AAAI)*, pages 90–97, 2000.
- [Scott 1992] W. Richard Scott, *Organizations: Rational, Natural and Open*, Prentice-Hall, 1992.
- [Sen and Luson 1997] Sen, A. and Luson, M. A new model for scheduling packet radio networks. In *Wireless Networks*, 3:71–82, 1997.
- [Silva 2002] Silva, O.; Garcia, A; Lucena, C. J. “The Reflective Blackboard Architectural Pattern for Developing Large Scale Multi-Agent Systems”. To appear in the Proceedings of the 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2002) at ICSE 2002, Orlando, USA, May 2002.
- [Smith and Davis 1983] R.G. Smith and R. Davis. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, pages 63–109, 1983.
- [So 1996] Y. So and E. Durfee, “Designing Tree-Structured Organizations for Computational Agents,” *Computational and Mathematical Organization Theory*, 2(3), 1996, 219-246.
- [Stentz 1995] Anthony Stentz. The focused D* algorithm for real-time replanning. In *Proceedings of the IJCAI*, 1995.

- [Tambe 1997] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
- [Tel 1991] G. Tel, *Topics in distributed algorithms*. Cambridge University press, pp. 27-31, 1991.
- [Verfaillie and Schiex 1994] Gerard Verfaillie and Thomas Schiex. Solution reuse in dcsp. In *AAAI94*, pages 307–312, 1994.
- [Wagner 2000] T. Wagner and V. Lesser. “Relating Quantified Motivations for Organizationally Situated Agents.” in Proceedings of ATAL 2000.
- [Yadgar 2002] O. Yadgar, S. Kraus and C. L. Ortiz, “Hierarchical organizations for real-time large-scale task and team environments”, in Proceedings of AAMAS, 2002.
- [Yadgar 2003] O. Yadgar, S. Kraus and C. L. Ortiz, “Information Integration Approach for Large-Scale Multiagent R.M.”, in *Communication in Multiagent Systems* (to appear), 2003.
- [Yokoo and Ishida 1999] Makoto Yokoo and Toru Ishida. *Multiagent Systems*, chapter 4. MIT Press, 1999.
- [Young-pa 1992] Y. So and E. H. Durfee, “A Distributed Problem-Solving Infrastructure for Computer Network Management.” *International Journal of Intelligent and Cooperative Information Systems*, 1(2):363-392,1992.
- [Yu 2002] E. Yu and L. M. Cysneiros, “Large-Scale Agent Systems: A World Modeling Perspective,” in SELMAS02, 2002.