Efficient Algorithms in Large-Scale Agent Systems

Osher Yadgar
Department of Computer Science

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University
Ramat Gan, Israel
May, 2005
This work was carried out under the supervision of Prof. Sarit Kraus, Department of Computer Science, Bar Ilan University.
Acknowledgments

I wish to express my gratitude to my advisor Prof. Sarit Kraus whose contribution to this thesis was tremendous. Setting an example throughout this long journey as a leading researcher was an inspiration to me. For her willing to share her wisdom, knowledge and experience, for her dedication, guidance and encouragement, for her understanding and acceptance I am thankful.
Contents

ABSTRACT ............................................................................................................................................. i

CHAPTER 1  INTRODUCTION .............................................................................................................1
  1.1. Preface......................................................................................................................................1
  1.2. Related Work ..........................................................................................................................7
  1.3. Structure of The Thesis .........................................................................................................12

CHAPTER 2  THE DISTRIBUTED DISPATCHER MANAGER (DDM) ............................................13
  2.1. The Coverage Density .........................................................................................................14
  2.2. Object Movement and Agent Measurements .....................................................................16
  2.3. Agents’ Measurements ........................................................................................................17
  2.4. Constructing an Information Map .......................................................................................19
  2.5. The DDM Hierarchy Architecture ......................................................................................20

CHAPTER 3  ARCHITECTURE ALGORITHM DESCRIPTION .....................................................23
  3.1. Sampler Capsule Generation Algorithm .............................................................................24
  3.2. Leader Localinfo Generation Algorithm ...........................................................................24

CHAPTER 4  ARCHITECTURE ALGORITHM COMPLEXITY .......................................................27

CHAPTER 5  LOAD BALANCE ALGORITHM ..................................................................................29
  5.1. Redirection of Agents Algorithm ........................................................................................34
  5.2. Redeployment of Agents Algorithm ...................................................................................39
List of Figures

Figure 1: Solving the problem and redistributing the agents ..................................................... 3
Figure 2: DDM hierarchy information flow diagram .............................................................. 22
Figure 3: Translating the physics notation to DDM .............................................................. 31
Figure 4: Target sampling by one Doppler ........................................................................... 56
Figure 5: Simulation of two sensors ...................................................................................... 58
Figure 6: The ResBy relation ............................................................................................... 69
Figure 7: Rectangle patrol movement pattern ................................................................. 73
Figure 8: Triangle patrol movement pattern ....................................................................... 73
Figure 9: Coverage density of the three experiment sets ..................................................... 74
Figure 10: Medium scale - Summary of methods .............................................................. 80
Figure 11: Medium scale - Target tracking percentage by settings .................................... 81
Figure 12: Medium scale - Target tracking average time by settings .................................. 81
Figure 13: Medium scale - Target detection percentage as a function of the communication noise ................................................................. 83
Figure 14: Medium scale - Target detection average time as a function of the communication noise ....................................................................................... 83
Figure 15: Medium scale - Tracking percentage as a function of the number of sensors ............................................................................................. 84
Figure 16: Large scale - Tracking average time as a function of the number of sensors .......... 85
Figure 17: Large scale - Tracking percentage by time in zone ............................................ 88
Figure 18: Large scale - Time to track distribution ............................................................ 89
Figure 19: Large scale - Tracking duration distribution ...................................................... 89
Figure 20: Large scale - Accurate tracked target percentage as a function of the number of levels ..................................................................................... 90
Figure 21: Large scale - Accurate tracking time as a function of the number of levels ....... 91
Figure 22: Large scale - Maximum agent process time as a function of the number of levels .................................................................92
Figure 23: Large scale - Bytes transferred as a function of number of levels .................................................93
Figure 24: Large scale - Average number of bytes received by a single agent as a function of number of levels ..........................................................................................93
Figure 25: Large scale - Accurate tracked target percentage as a function of non-functioning samplers .............................................................................................................94
Figure 26: Large scale - Accurate tracked target percentage as a function of non-functioning first level leaders ..............................................................................................................95
Figure 27: Large scale - Accurate tracked target percentage of patrol as a function of lost communication messages between samplers and leaders .................................96
Figures 17* and 18*: Large scale - Tracking percentage by time in zone and by Time to track distribution .................................................................................................................97
Figures 19* and 20*: Large scale - Tracking duration distribution and Accurate tracked target percentage as a function of the number of levels .................................................98
Figures 21* and 22*: Large scale - Accurate tracking time as a function of the number of levels and Maximum agent process time as a function of the number of levels .................................................................................................................................99
Figures 25* and 26*: Large scale - Accurate tracked target percentage as a function of non-functioning samplers and Accurate tracked target percentage as a function of non-functioning first level leaders .................................................................................99
Figure 28: Very large scale - Basic settings with load balance - average target tracking - 10 sec ..........................................................................................................................104
Figure 29: Very large scale - Basic settings with load balance - average target tracking - 1 hour ..........................................................................................................................105
Figure 30: Very large scale - Basic settings with load balance - average target tracking - 3 Hours ..........................................................................................................................105
Figure 31: Very large scale - Basic settings with load balance - average target tracking comparison ..........................................................................................................................106
Figure 32: Very large scale - Basic settings with load balance - time to track comparison

Figure 33: Very large scale - Basic settings without load balance – average target tracking - 10 sec

Figure 34: Very large scale - Basic settings without load balance – average target tracking - 1 Hour

Figure 35: Very large scale - Basic settings without load balance – average target tracking - 3 Hours

Figure 36: Very large scale - Basic settings without load balance – average target tracking comparison

Figure 37: Very large scale - Basic settings without load balance - time to track comparison

Figure 38: Very large scale - Basic settings – overall average target detection comparison

Figure 39: Very large scale - Basic settings – overall time to track comparison

Figure 40: Very large scale - Basic settings – LB time to track distribution

Figure 41: Very large scale - Basic settings – NLB time to track distribution

Figure 42: Very large scale - Sensor population: LB average target detection

Figure 43: Very large scale - Sensor population: NLB average target detection

Figure 44: Very large scale - Sensor population: LB versus NLB Average Target Detection

Figure 45: Very large scale - Sensor population: LB versus NLB time to track

Figure 46: Very large scale - Sensor range of interaction: LB versus NLB average target detection

Figure 47: Very large scale - Sensor range of interaction: LB versus NLB time to track

Figure 48: Very large scale - Moving/Sensing time: LB versus NLB average target detection

Figure 49: Very large scale - Moving/Sensing time: LB versus NLB time to track
Figure 50: Very large scale - Hierarchy levels: LB versus NLB average target detection
........................................................................................................................................124
Figure 51: Very large scale - Hierarchy levels: LB versus NLB time to track...............124
Figure 52: Very large scale - Movement pattern: LB versus NLB average target detection
........................................................................................................................................125
Figure 53: Very large scale - Movement pattern: LB versus NLB time to track..........126
Figure 54: Very large scale - Tracking percentage as a function of the coverage density
........................................................................................................................................127
Figure 55: Very large scale - Alternative settings for the basic settings...............127
Abstract

In everyday life we often encounter devices forming large scale environments. Such environments could be the Internet, a cellular network, an array of fire detection sensors, an array of solar receptors and so on. Recent technology has made small low cost devices available to build sensor networks [21, 30, 32, 36, 75, 27, 64]. As technology advances opportunities arise to form large scale cooperative systems in order to solve larger problems in an efficient way. As more large scale systems are developed there is a growing need to (i) measure the scale of a given large scale problem; (ii) compare them to other large scale systems in order to extract a suitable solution; (iii) predict the performance of the solution and (iv) derive the value of each system property from the desired performances of the solution, the problem constraints and the user’s preferences.

The following research proposes a novel system term, the coverage density, to define the hardness of a large scale system. This term, can be used to compare two different scales of large scale systems in order to find similar systems and suitable solutions. Given a coverage density of a system one may predict the solution performances and use it jointly with the preferences and the constraints to derive the value of the system's properties.

In this work we suggest an efficient system architecture to manage large scale environments and demonstrate the importance of the coverage density term. This system architecture is a generic solution for a variety of large scale agent problems. These problems may be, for instance, surveillance tasks in a closed compound, Internet data mining, cellular network operation, space and planet exploration, Tsunami and earthquake alert systems, aerial control systems and so on.

To adopt the proposed architecture there is a need to supply only three simple domain related functions, while the remaining algorithms are kept in their generic form. We show that the proposed architecture is a robust, fault tolerant, efficient and scalable model. Moreover, it is a cost-effective solution since its information fusion mechanism can utilize faulty, noisy and very cheap sensors.

As an enhancement of the proposed architecture the use of a load balancing mechanism in cases of uneven distribution of tasks is also proffered. A novel load balancing
algorithm which can be implemented by the suggested system architecture is presented empirically demonstrating a tremendous improvement in performance.

In this study we tested the performance of the suggested large scale solutions and the load balance algorithm through extensive experimentation in a simulated environment involving thousands of sensors. A total of 50 years of CPU time were logged, examining hundreds of thousands of different agents and goals. The profound scale of these experiments establishes the efficiency of the suggested large scale solution and the importance of the coverage density as a large scale system measuring tool.
Chapter 1

Introduction

1.1. Preface

In December 2, 2004 tsunami waves hit the shores of Sri Lanka, India, Indonesia and Thailand causing a loss of 300,000 lives and a tremendous tragedy for millions. Using current technology, sensors in the ocean could have sensed the creation and advancement of tsunami waves. A well organized and managed network of such sensors using wireless communication could have produced an alarm which would have alerted control centers spread along the shores of Sri Lanka, India, Indonesia and Thailand. This alarm could have saved the lives of many of the victims.

Wireless sensor networks benefit from micro-electro-mechanical systems (MEMS) technology advances [33]. These advances have facilitated the development of low cost simple wireless sensors. Combining the information gathered from thousands of such sensors is a very difficult problem [70]. However, solving this problem may lead to an efficient way of producing global information. As we have witnessed such information could save lives.

Large scale agent systems focus on the behavior of multi agent systems with many agents. As this is a relatively new research area, the number of agents needed to consider a multi agent system as a large scale agent system has not yet been defined in literature [81]. In some cases thousands of agents are considered to be large scale agent systems [10, 24, 61], while in other cases hundreds comprise such systems [15, 12, 62, 63] and there are even cases where only dozens of agents are considered to be large scale systems [37]. This work involves all three types of large scale systems, with focus on systems with thousands of agents. Furthermore, we suggest that large scale agent systems should be measured relative to their context, i.e. the size of their problem space, and not simply by the number of participating agents.

A novel multi-agent solution to the problem of resource management in very large-scale task and resource environments is presented in this thesis. The study focuses on domains of application in which resources are best modeled by autonomous mobile agents, in
which each can decide to take on new tasks in their immediate environment. Since agents are mobile, they can be redirected to other areas where resources are needed. However, since no single agent has global knowledge regarding the distribution of tasks and resources, local information from agents must be pooled to obtain a more accurate understanding of the global situation.

Central to our approach is a hierarchical agent organization for distributing the load of computation and for reducing inter-agent communication. Distribution of computation load and reduction of agent communication is critical for allowing a scale of up to thousands of agents [48, 15]. It has been shown [53, 60] that computation load is the main concern when maintaining the scalability of a system. It has also been shown [25, 50, 39, 74] that the overhead of communication is a likely cause of scalability failure.

The multi agent system community has extensively studied the problem of distributed vehicle monitoring and tracking as an example for distributed resource allocation [67, 80]. We also describe and examine a domain involving sensor webs that jointly perform a surveillance task. This domain can be generalized to correspond to many agent domains with objects that appear in the environment acting as agent’s tasks. Suitable domains may be surveillance tasks in a closed compound, Internet data mining, cellular network operation, space and planet exploration, Tsunami and earthquake alert systems, aerial control systems and more. In order to analyze problems in these domain types, the problem solving process is divided into three stages:

i. A situation assessment stage in which information processed from individual agents is extended with causal knowledge about likely object behaviors and then combined to form a global situation assessment.

ii. A resource load balancing stage in which agents perform a balancing process to issue (re-)distribution orders to subordinate agents.

iii. A resource deployment stage in which agents are (re-)deployed for better task management.

Specifically, the thesis approach is as follows: Agents are grouped into agent groups, each with a distinguished agent group leader. The agent groups might be assigned to a specific localized sector. A sector may be a geographic area of interest or logical units regardless of
their physical location. The groups themselves are combined into larger groups. Communication is restricted to flow only between an agent (or agent group leader) and its group leader. State information from individual agents is sent to agent group leaders and sector assignments are sent from the group leader to the agents. Each individual agent can position itself within an assigned sector depending on the tasks (objects) that it detects in its local environment. In this model, therefore, resources are not directly allocated to tasks but are rather distributed to sectors where it is believed they are most needed: the sector leader does not need to know exactly which agent is going to take on a particular task.

Schematically the problem and its solution can be described as follows:

Figure 1: Solving the problem and redistributing the agents

The described problem presents a number of difficult challenges:

i. Data association and object identification information must be connected with the task state measurements from one time point to the next. In other words, large amounts in the scale of thousands of data must be analyzed across each time unit and connected to historical data.

ii. Local information obtained by an agent is incomplete and uncertain and must be combined with other agents’ local information to improve the assessment.
iii. Computing the information map and tracking objects must be accomplished in real time or in soft real time. This is one reason for giving individual agents the flexibility to act more or less autonomously within their sector: agents can react to nearby targets.

In this thesis we will describe the Distributed Dispatcher Model (DDM), a system embodying these ideas. DDM is designed for efficient coordinated resource management in systems consisting of thousands of agents; the model makes use of hierarchical group formation to restrict the degree of communication between agents and to guide processes in order to very quickly combine partial information to form a global assessment. Each level narrows the uncertainty based on the data obtained from lower levels. We show that the hierarchical processing of information reduces the time needed to form an accurate global assessment.

It is also shown that in some cases naïve distribution of thousands of agents will not be efficient; thus requiring a load balancing mechanism. A load balancing mechanism is suggested for the hierarchic architecture of thousands of agents. This load balancing mechanism dynamically balances the ratio between agents and goals throughout the controlled area. The DDM strives to balance the ratio between agents and goals in each hierarchy level from top down. According to the load balance mechanism each level directs only its immediate subordinate level. The directed level, afterwards, directs its own immediate subordinate level and so on and so forth.

In the simulation models a suite of Doppler sensors are used to form an overall assessment about the location of targets moving at a steady velocity. A Doppler sensor is a radar based on the Doppler effect. A Doppler sensor provides only partial information about a target, in terms of an arc on which a detected target might be located and the velocity towards it, that is, the \textit{radial velocity} [64]. Given a single Doppler measurement, one cannot establish the exact location and velocity of a target [81]; therefore, multiple measurements must be combined for each target. This problem was devised as a challenge problem by the DARPA Autonomous Negotiating Teams (ANTS) program to explore realtime distributed resource allocation algorithms in a two dimensional geographic environment [19, 4, 3]. The ANTS program uses Doppler sensors that may activate its beam in three different directions.
According to the ANTS specific Doppler radar, only one direction may be activated at a time. The orientations of the beams are 0, 120 and 240 degrees.

The DDM hierarchical architecture is compared to other architectures and the results show that situation assessment is faster and more accurate in DDM. The results of the comparison also show that DDM can achieve improved results while only using a low volume of possibly noisy communication. In addition, the consequences of activating a load-balancing algorithm as opposed to naïve distribution of agents were examined.

The results of this work show that hierarchy models such as DDM manage large-scale agent systems efficiently. The suggested DDM may be used for many large-scale agent management problems, such as those detailed above. We will show that only three portable functions depend on the specific domain characteristics. Only these functions will have to be adjusted, with all other algorithms remaining domain independent. We will also show how to apply a load balancing mechanism in large-scale agent environments and the benefits derived thereof. The experiments allow us to examine the question of how to determine the number of levels of the DDM hierarchy in a large-scale system. The results reveal 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.

Subsequently, we conducted a comprehensive study to evaluate the hierarchical solution. In this part we demonstrated the advantages of DDM and the load balance algorithms using a simulator that was developed especially for this purpose. Three scales of agent systems were studied and compared: medium scale agent systems, large scale agent systems and very large scale agent systems. While comparing these systems we show the importance of the coverage density to measure the scale of a system and to predict the performances in a given setting. We conclude by discussing the major contribution of the DDM solution to the large-scale agent system challenges in terms of capability, accuracy, efficiency, cost-effectiveness, robustness and fault tolerance.
In summation, this work differs from other works that have been done in this field in reference to the following areas:

**Generalization:** In this work a generic architecture for large scale agent systems is presented. This hierarchic architecture is domain independent. It is only necessary to specify three very simple domain related functions and integrate them into the specified placed in the algorithm.

**Extensive empiric results:** 150 different pc computers running XP, windows 2000 and Linux operation systems were used to simulate hundreds of scenarios for four consecutive months. A total of 50 years of CPU time were logged, examining hundreds of thousands of different agents and goals.

**Load balancing:** A hierarchic load balancing mechanism to balance the agents and object ratio across the controlled zone is presented. This load balancing algorithm, described in chapter 5, was inspired by physical fields of potential.

**Coverage density:** This thesis presents a novel term to measure and compare large scale agent systems. We suggest that the degree of a large scale agent system be considered in a relative means and not only by absolute figures, such as the number of agents.

**Sensor:** DDM supports mobile sensor agents. These agents may physically or virtually move from one state to another. To take part in the suggested DDM solution and the load balancing mechanism, 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 show that as opposed to previous similar work [6, 8] a single agent using one Doppler radar can track an object by taking multiple sequential measurements and combining them. In our example, 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 requirement between agents is lessened and, in particular, synchronization between agents is unnecessary.
Noise: we show that redundancy of agents may compensate for faulty and noisy sensor measurements and for communication loss. This fault tolerance capability is important in large scale agent systems.

1.2. Related work

The idea of combining partial local solutions into a more complete global solution can be traced back to early work on the distributed vehicle monitoring testbeds (DVMT) [40]. DVMT also operated in a domain of distributed sensors that tracked objects. However, the algorithms for supporting mobile sensors and for the actual specifics of the Doppler sensors themselves are novel to the DDM system. Within the DVMT, Corkill and Lesser [18] investigated various team organizations in terms of interest areas which partitioned problem solving nodes according to roles and communication, but they were not initially hierarchically organized [76, 83]. Wagner and Lesser examined the role that knowledge of organizational structure can play in control decisions [79]. DDM differs in its organization used to dynamically balance computational loads and also in its algorithms which support mobile agents.

The benefits of hierarchical organizations have been argued by many [11, 55, 56, 35, 71]. Organization theorists emphasize control of resources through hierarchy structure. These works focus on how managers can effectively allocate resources or workers in lower levels of the hierarchy. In some of their work we can find managers at different levels deciding how to allocate resources to workers in lower levels in the hierarchy [51, 68, 69, 56, 57, 58, 59]. These works can generally be categorized as Top-Down decomposition and Bottom-Up aggregation of hierarchical strategies for solving tasks. In a Top-Down decomposition the individual or highest group may decide to decompose tasks into subtasks in a hierarchic form. Each task can in turn be decomposed and passed down the hierarchy while the combination of the outcome achieves the original goal. So and Durfee draw on the contingency theory to examine a variety of hierarchical organizations' benefits; they portray a hierarchically organized network monitoring system for task decomposition and they also consider organizational self-design [86, 85].
In a Bottom-Up aggregation subtasks are assigned to the lower level members of a hierarchy. The aggregation in a hierarchic form of the subtasks results in the overall goal [31, 78, 9, 17, 66, 77, 49]. In our work both Top-Down decomposition and Bottom-Up aggregation approaches are used. The Bottom-Up aggregation approach is used to collect information about objects from different subsections to form a global information view about them. Thus data fusion processing is made by every leader to fuse data segments from different sources and historic data. Such distributed data fusion processing reduces the computation load. It also reduces the amount of data passed between two leaders when data segments are fused. The Top-Down decomposition approach is used to balance the sensing task load among sensors. According to their information view and instructions from their leader, leaders may calculate the sensors per objects imbalance in their controlled subzones and instruct their sub-leaders to pass agents from one subzone to another. This recursive approach reduces the commutation load since every leader solves the imbalance only among its immediate sub-leaders.

The term, sensor coverage, is traditionally used to denote the effectiveness of a sensor network [45, 1, 44, 43, 42, 73, 38]. Previous studies have shown that increasing the sensor coverage increases the number of tracked objects. Gage [22] defines the coverage as a "spatial relationship which adapts to specific local conditions to optimize the performance of some function". Gage states that in many systems of a large number of robots, i.e. more than 100, the desired group behavior is the maintenance of this spatial relationship. Gage distinguishes between three basic types of coverage behaviors: blanket coverage, where the objective is to achieve a static arrangement of sensors that maximizes the detection rate of targets appearing within the coverage area; barrier coverage, where the objective is to achieve a static arrangement of sensors that minimizes the probability of undetected targets passing through a barrier; and sweep coverage, where the objective is to move a group of elements across a coverage area to balance between maximizing the detection rate and minimizing the number of missed detections. Gage specifies that a sweep is equivalent to a moving barrier. Optimization of sensor placement has led to a centralized approach. According to this approach static sensors are placed in advance to maximize the coverage of the sensor network [34]. In this work we show that in a dynamic large scale environment in which the number and location of objects that should be tracked cannot be predetermined, dynamic approaches
may produce better results. Such a dynamic approach is presented by Kian et al. [37]. Their work portrays a task allocation scheme via self organizing swarm coalitions for distributed mobile sensor network coverage. They use ant behaviors as motivation to self regulate the regional distribution of sensors and thus increase the coverage of a sensor network. They show that other strategies such as static sensor placement, potential fields and auction based negotiation achieves less coverage and are less flexible to environment changes.

While Kian et al. refer to dozens of robots in their multi agent system, our system handles thousands. In such large scale environments, a fully distributed architecture such as Kian suggests cannot produce global information in real time because of communication bottleneck between the distributed robots and the center. Moreover, in contrast to Kian’s work, we use the term of coverage density instead of the traditional term of coverage. We show that the sensor coverage should be considered along with other properties such as the change of coverage with time.

A paradigm for robust, scalable and energy efficient data dissemination for a sensor network is presented by Intanagonwiat and Estrin [12, 13, 20]. They named their data centric paradigm direct diffusion. According to the paradigm, sensors sensed data is organized in attribute-value pairs. A query for data may be sent by network nodes and will be “drawn” down towards that node. The paradigm of direct diffusion is aimed to satisfy queries and consists of the following properties: interests, data, messages, gradients and reinforcements. An interest represents a query that specifies what a user wants. A sensing task is disseminated throughout the sensor network as an interest. Gradients, then, are set within the network to “draw” events. The direction of the gradient is set toward the neighboring node from which the interest is received. The gradient results in a flow of events towards the originators of interests along multiple gradient paths. The sensor network reinforces one, or a small number of these paths. This paradigm is not suitable to acquire global information about objects in large scale systems in real time since nodes neighboring to the command center node are likely to become bottlenecks.

In our work gradients are used to describe the imbalance between neighboring subzones. There is only one gradient between two neighboring subzones. In contrast, the use of the direct diffusion paradigm results in the creation of gradients proportional to the number
of queries. This amount of communication exchange makes this solution problematic when scaling in very large systems.

An approach that takes advantage of mobile components in wireless sensor network systems is presented by Kansal [2]. This approach exploits mobility to develop a fluid infrastructure; mobile components are deliberately built into the system infrastructure to solve networking problems. In contrast to Kesal’s work, the large number of mobile sensors is an inherent part of the system and should not be used only for networking purposes.

Alternative approaches to realtime distributed resource allocation are explored within the ANTS program [18, 41, 14, 58, 59]. Most of these other approaches assume that agents are stationary. Vincent et al. use a limited team organization, assigning agents to sector managers [65]. Each sector manager is responsible for fusing information for tracking. A hierarchical organization serves to limit communication among agents and to provide a measure of fault tolerance; if a sector manager is disabled, another can fill in.

Another solution to the ANTS problem was introduced by Leen-Kiat and Costas [41]. They used measurements of three Doppler sensors at the same time and intersected the resulting arcs of each. 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 must be used to gain that synchronization. In addition, communication is required for scheduling which agent will take measurements when. We offer a different way to use uncertain measurements and focus on the combination process of partial and local information.

In their work, Yu and Cysneiros [26] 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 consider these challenges. We propose a way to solve them and prove our solution efficiency.

Silva et al. present the Reflective Blackboard architectural pattern for large-scale systems [54]. This pattern is the result of the composition of two other well-known architectural patterns: the Blackboard pattern and the Reflection pattern. They separate the 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.

Achieving scalability through hierarchic architecture has been extensively studied for years. Lumer and Huberman [5, 23] used a hierarchy of processors to gain stability in a system with many processors. They applied mathematical techniques based on chaos theory to analyze the distributed computer networks and examined a load balancing mechanism in which heavy loaded processors delegated tasks to lightly loaded processors. They found that for more than a certain number of processors (i.e. 21 in a typical case) the system was unstable. They used a hierarchic architecture solution to gain stability and scalability. According to this architecture processors are grouped hierarchically into clusters. Tasks are delegated between processors in the same cluster and rarely between possessors from different clusters.

Tel studied a network tree with n processors providing communication between every pair of processors with a minimal number of links (n-1) [29]. 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 better communication complexity. However each node has more computations to do and thus can become a bottleneck. A failure of a node will divide the tree into a higher number of unconnected subsets. In our work we study the relation of the number of levels in a hierarchical structure with the performance and then provide suggestions of how to choose the right number of levels.

Horling et al. [7] describe architecture for a multi agent large scale distributed sensor network. The controlled zone is first divided by the agents into a series of sub-zones, each a non-overlapping, identically sized, rectangular portion of the available area. Agents may manage a sub-zone, track targets and produce sensor data and they are arranged in a grid pattern. The architecture is based on a hierarchy stricture of only two levels. Track managers obtain their local information from their original sub-zone manager. They may also interact directly with other sub-zones and track managers. Therefore they do not follow a fixed chain of command. While testing the scalability of their architecture they varied the number of targets from 1 to 24 and the number of sensors from 1 to 81.
The DDM architecture is different from Horling's architecture in reference to the following aspects:

i. DDM has been tested for a much larger community of sensors and targets (thousands versus dozens) and has proven its scalability;

ii. DDM uses mobile sensing agents versus sensors in a grid pattern. As shall be demonstrated, this dynamic capability produces better results;

iii. DDM does not restrict the number of levels and may increase it to support scalability;

iv. DDM has only two types of agents: sensing and group leading agents;

v. Although sensing agents may move autonomously in a given sub-zone, DDM uses a strict chain of command in which sensors may change their manager.

1.3. Structure of the thesis

The research presented introduces an efficient system architecture to manage large scale environments, the coverage density term to define the hardness of a large scale system and a load balancing mechanism to balance the load of agent per tasks in problem space. In chapter 2 we present the distributed dispatcher manager architecture as an efficient system architecture for large scale agent systems. We then describe the properties involved in such large scale systems and define the coverage density. We proceed, in chapter 3, by detailing the algorithms of the agents forming the distributed dispatcher manager while in chapter 4 we analyze the distributed dispatcher manager algorithm's complexity. Next, in chapter 5, we present the hierarchic load balance algorithm that balances the ratio between agents and tasks in the problem space. The complexity analysis is detailed in chapter 6. Following DARPA's ANTS program, an implementation of the distributed dispatcher manager is presented in chapter 7. This example demonstrates how to implement the domain dependent parts of the algorithms. A profound set of experiments is presented in chapter 9 along with their analyzed results. We conclude, in chapter 9, with a summary of our thesis and our insights.
Chapter 2

The Distributed Dispatcher Manager (DDM)

In general, this thesis discusses the complexities that arise when distributed agent networks are scaled up to thousands of information collecting agents, i.e. sensing agents, and thousands of goals. A model recommended for effectively managing such networks, called the Distributed Dispatcher Manager (DDM), is described in this section.

DDM organizes the sensing agents in teams, each with a distinguished team leader agent. A team is assigned to a specific sector of interest. Each such agent can act autonomously within its assigned sector of interest 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 agent. Each team leader is provided with an algorithm to integrate information obtained from its team members.

Each individual information-collecting agent can extend its local information through the application of causal knowledge. The causal knowledge may be inaccurate and noisy and therefore it only constrains the set of possible paths that could be associated with a collection of data measurements.

More abstractly, such problems and their solution will be modeled in the following way. A resource management problem, $\text{MP}$, will be defined as a tuple, $\text{MP}=\langle O, S, T, A, Sa, G, g, \text{Comm}, \sigma, \text{ResBy}, Y \rangle$, such that

$O$ : is a set of tasks or objects;
$S$ : is a set of objects or tasks states;
$T$ : is a set of integer times;
$A$ : is a set of agents;
$Sa$ : is a set of agent states;
$G$ : is a set of groups;
$g$ : is an assignment of agents and groups to groups such that $g: A \cup G \rightarrow G$;
$\text{Comm}$ : is a binary relation indicating a communication link between agents such that $\text{Comm} \subseteq A \times A$;
\( \sigma : \) is the actual states of tasks at a given time, \( \sigma : T \times O \rightarrow S; \)

\( \text{ResBy} : \) is a causal relation, \( \text{ResBy} \subseteq S \times T \times S \times T, \) that constrains the evolution of object states such that \( \text{ResBy}(s_1, t_1, s_2, t_2) \) iff \( s_2 \) at \( t_2 \) can follow \( s_1 \) at time \( t_1; \)

\( y : \) is an agent deploy order specifying an exact future time and state for a single agent. The agent deploy order will be defined as \( y = (t, sa); \)

\( Y : \) is a set of agent deploy orders.

**General notations**

**Object state function** \( f_o : \) For finding a solution to MP let us consider the notion of an object state function \( f_o : T \rightarrow S \) that associates its state change over time with an object \( o. \)

**Information map** \( I : \) An information map, \( I, \) will then be defined as a set of object state functions.

**Solution**, \( \Sigma : \) A solution, \( \Sigma, \) to a given MP, will be written \( \Sigma(MP) \), such that \( \Sigma(MP) \subseteq I, \) iff each object in \( O \) is captured in an actual object state function in \( \Sigma(MP). \)

An extension of this formalization will be given in subsequent sections.

### 2.1. The coverage density

Many properties influence the scale and the hardness of a given problem. When designing a solution for a large scale problem the different aspects of the system's properties should be weighted. Properties, such as the number of sensors and the quality of the sensors may be related to each other. For instance, given a limited budget, the system designer should consider whether to use many cheap sensors or a small number of expensive sensors.

A classifying figure, the coverage density, is proposed to classify large scale agent problems and to predict the performance of different property configurations. This predicting tool can be used to design the system in order to meet imposed constraints such as budget limitations, battery supply or technological constraints.
The coverage density defines the time it takes to cover an area equal to the size of the controlled zone. The following propositions formalize the way to find the coverage density of a given system.

**Proposition 1:**
Let \( a_i, \omega(t) \) be the area covered by the sensor of agent \( a_i \) at a given time \( t \). The agent may detect all objects in this area at time \( t \).
The measurement units of the area are square meters.

**Proposition 2:**
Let agent coverage, \( \omega_i \), be the average area covered by the sensor of agent \( i \) such that
\[
\omega_i = \frac{\int a_i, \omega(t) \cdot dt}{\int dt}
\]
The measurement units of the agent coverage are \( \text{square meters} \ \text{second}^{-1} \).

**Proposition 3:**
Let total coverage, \( A, \omega \), be the average area covered by the sensors of all the agents such that
\[
A, \omega = \sum \omega_i
\]
The measurement units of the total coverage are \( \text{square meters} \ \text{second}^{-1} \).

**Proposition 4:**
Let \( Z \) be the size of the controlled area.

**Proposition 5:**
Let coverage density \( \rho \) be the total coverage divided by the the size of the controlled area such that
\[
\rho = \frac{A, \omega}{Z}.
\]
The measurement units of the coverage density, \( \rho \), are \( \frac{1}{\text{second}} \).

The coverage density figure depicts the amount of the controlled area that can be covered. However, there may be an overlapping of agent coverage such that, for example, a coverage density of 100\% does not necessarily reflect full coverage of the controlled area. A coverage density value of more than 100\% implies a certainty that there is an overlapping of agent coverage. The coverage density term may be easily adopted in other large scale fields. Using other domain measurement units instead of Euclid’s space leads to the same definition.

### 2.2. Object movement and agent measurements

The ResBy function is designated to capture constraints. The relation ResBy holds for two object-states, \( s_1 \) and \( s_2 \), and two time points, \( t_1 \) and \( t_2 \), where \( t_2 \geq t_1 \). If it is possible that a state of an object is \( s_1 \) at \( t_1 \), then state \( s_2 \) could be at time \( t_2 \). ResBy should also satisfy the following constraints:

Let \( t_1, t_2, t_3 \in T \) and \( s_1, s_2, s_3 \in S \), \( t_1 < t_2 < t_3 \) such that

(i) if \( \text{ResBy}(< t_1, s_1 >, < t_2, s_2 >) \) and \( \text{ResBy}(< t_2, s_2 >, < t_3, s_3 >) \) then

\[
\text{ResBy}(< t_1, s_1 >, < t_3, s_3 >)
\]

(ii) if \( \text{ResBy}(< t_1, s_1 >, < t_2, s_2 >) \) and \( \text{ResBy}(< t_1, s_1 >, < t_3, s_3 >) \) then

\[
\text{ResBy}(< t_2, s_2 >, < t_3, s_3 >)
\]

(iii) if \( \text{ResBy}(< t_1, s_1 >, < t_3, s_3 >) \) and the \( \text{ResBy}(< t_2, s_2 >, < t_3, s_3 >) \) then

\[
\text{ResBy}(< t_1, s_1 >, < t_2, s_2 >)
\]

(iv) if \( \text{ResBy}(< t, s_1 >, < t, s_2 >) \) then \( s_1 = s_2 \).

The constraints (i)-(iii) on ResBy restrict the way the state of an object may change over time. They refer to three points of time \( t_1, t_2, t_3 \) in an increasing order and to the
possibility that an object was at state $s_1$, $s_2$, and $s_3$ at these time points, respectively. If the object was in these states at the corresponding times then $s_2$ at $t_2$ should be a result of $s_1$ at $t_1$, i.e. $ResBy(<t_1, s_1 >, <t_2, s_2 >)$. Similarly $ResBy(<t_2, s_2 >, <t_3, s_3 >)$ and $ResBy(<t_1, s_1 >, <t_3, s_3 >)$. The constraints indicate that it is suffice to ascertain that two out of the three relations hold to verify that the object was really at $s_1$ at $t_1$, $s_2$ at $t_2$, and $s_3$ at $t_3$. In other words, if two of the three relations hold, the third one must also be true. The last constraint (iv) is based on the fact that an object cannot be in two different states at the same time.

2.3. Agents’ measurements

Each agent is capable of taking measurements or sampling its nearby environment. Object measurements provide only partial information on object-states and may be incorrect. When an agent takes measurements its agent state is referred to as the viewpoint from which a particular object state was measured. It is assumed that there is a function $PosS$ that given $k$ consecutive measurements taken by the same agent, up to time $t$, returns a set of possible states, $S' \subseteq S$, for an object at time $t$ where exactly one $s \in S'$ is the right object state and there is an $m \geq 1$ such that $|S'| \leq m$.

**Definition 1:** A path, $p$, is a sequence of triples $<<t_1, sa_1, s_1>, ..., <t_n, sa_n, s_n>>$ where

$t_i \in T, s_i \in S, sa_i \in Sa$ and for all $1 \leq i < n$, either

$t_i < t_{i+1}$ and $ResBy(<t_i, s_i >, <t_{i+1}, s_{i+1} >)$ is true or

$t_i = t_{i+1}$, $sa_i \neq sa_{i+1}$ and $s_i = s_{i+1}$.

Each path represents an object’s discrete state change over time as measured by sampling-agents in states, $sa_1 ... sa_n$. Constraint (i) considers the case where two points in the path capture the change of the state of the object from $s_i$ at time $t_i$ to $s_{i+1}$ at time $t_{i+1}$. In this
case, where the path specifies the way the state was changed, \( \text{ResBy}(<t_i, s_i>, <t_{i+1}, s_{i+1}>) \) must hold, i.e. the object could be at \( s_i \) at \( t_i \) and then at \( s_{i+1} \) at \( t_{i+1} \). On the other hand, constraint (ii) considers the case of two points \( <t_i, sa_i, s_i>, <t_{i+1}, sa_{i+1}, s_{i+1}> \) on the path that do not capture a change in the object’s state but rather two different observations of the object. That is, the object was at a given state \( s_i \) at time \( t_i \), but was observed by two agents. The two agents were, of course, in different states, and this is captured by the constraint that \( sa_i \neq sa_{i+1} \).

A path often only consists of a very few states of an observed object. However, an agent would like to infer the state of the object at any given time from a path function. This is formalized as follows.

**Definition 2:** An object state function \( f_{\pi_i, \pi_e} \), with respect to two path points \( \pi_i =<t_i, sa_i, s_i>, \pi_e =<t_e, sa_e, s_e> \) where \( t_i \leq t_e \), associates an object state with each time point (i.e. \( f_{\pi_i, \pi_e} : T \rightarrow S \)) such that

i. \( f_{\pi_i, \pi_e}(t_i) = s_i \) and \( f_{\pi_i, \pi_e}(t_e) = s_e \)

ii. \( \forall t_1, t_2, t_1 < t_2 \ \text{ResBy}(<t_1, f_{\pi_i, \pi_e}(t_1)>, <t_2, f_{\pi_i, \pi_e}(t_2)>) \).

An object state function represents an object state change over time points in \( T \) with respect to two path points. To move from a path to an associated function, it is assumed that there is a function \( \text{pathToFunc}: P \rightarrow F \) such that given a path \( p \in P \),

\[ p = <<t_1, sa_1, s_1>, ..., <t_n, sa_n, s_n>>, \text{ if } f_{\pi_i, \pi_e} = \text{pathToFunc}(p) \text{ then, } \pi_i =<<t_1, sa_1, s_1>>, \]

\[ \pi_e =<<t_n, sa_n, s_n>, \forall t_i, f_{\pi_i, \pi_e}(t_i) = s_i. \]

An object state function and its path are used interchangeably throughout the remainder of the paper.
2.4. Constructing an information map

DDM uses partial and local information to form an accurate global description of the changes in objects over time. The DDM model can be applied to many commands, control and intelligence problems by mapping the DDM entities to the domain entities. As pointed out earlier, the goal of DDM is to construct an information map.

Definition 3: An information map, infoMap, is a set of object state functions

\[ \langle f^{1}_{\pi_{i}, \pi_{j}}, \ldots, f^{h}_{\pi_{i}, \pi_{j}} \rangle \text{ such that for every } 1 \leq i, j \leq h \text{ and } t \in T, f^{i}_{\pi_{i}, \pi_{j}}(t) \neq f^{j}_{\pi_{i}, \pi_{j}}(t) \]

Intuitively, infoMap represents the way that the states of objects change over time. The condition on the information map specifies the assumption that two objects cannot be at the same state and time. Because each agent has only partial and uncertain information of its local surroundings an agent may need to construct the infoMap in stages. In some cases, an agent might not be able to construct the entire infoMap. The process of constructing the infoMap will use various intermediate structures.

As mentioned above, to capture the uncertainty associated with sensed information, each sampled object is associated with several possible object states. We introduce the notion of a capsule that represents a few possible states of an object at some time as derived from measurements taken by an agent in a given state.

Definition 4: A capsule is a triple of a time point, a sampler agent state and a sequence of up to \( m \) object-states, i.e., \( C = \langle t, sa, \{s_{1}, \ldots, s_{l}\} \rangle \), where \( t \in T, sa \in Sa, s_{i} \in S, 1 \leq m \). The set of all possible capsules is denoted \( C \). Capsules are generated by the sampling agents using the domain dependent function \( PosS \) and \( k \) consecutive samples.

The assessment problem discussed earlier corresponds to the problem of how to choose the right state from every capsule. It is impossible to determine which state is the correct state using only one viewpoint: measurements from one viewpoint can result in up to
m object states, each of which could correspond to the correct state. Therefore, capsules from
different viewpoints are needed. A different viewpoint may correspond to a different state of
the same sampling agent or of different sampling agents. To choose the right object state
from each capsule state, different capsules are connected using the ResBy relation to form a
path. Each of these paths is evaluated and those with the best probability are chosen to
represent the most likely sequence of object state transitions to form state functions.

**Definition 5:** *localInfo* is a pair of *infoMaps* and a set of capsules, <infoMap,
unusedCapsules> where unusedCapsules=<c_1,...,c_m> s.t. for all 1 \leq i \leq m and for all 1 \leq j \leq l
and \( C_i = <t_i, sa_i, \{s_{i1},...,s_{ij}\}> \) and for every \( f_{\pi_i,\pi_j} \in \text{infoMap} \) \( f_{\pi_i,\pi_j}(t_i) \neq s_{ij} \).

At any time, some capsules can be used to form object state functions that have a high
probability of representing objects. These functions are recorded in infoMap; they are
referred to as *accurate representations*. The remaining capsules are maintained in the
unusedCapsules set and are used to identify state functions. That is, the condition of
definition 3 intuitively states that an object associated with a function \( f_{\pi_i,\pi_j} \) was not
constructed using one of the measurements that was used to form the capsules in the
unusedCapsules set.

### 2.5. The DDM hierarchy architecture

In a large-scale environment many capsules may have to be linked from each area.
Applying the ResBy relation many times can be time consuming. However, in many large
scale agent systems [53, 7] localization is very important. In those cases many of the relevant
interactions and information of a goal resides close to it. Following the localization behavior
of such systems, there is only a low probability that capsules created based on measurements
taken far away from one another will be related. Therefore, it is logical to distribute the
solution. The DDM hierarchical structure guides the distributed construction of the global
infoMap. The lower level of the hierarchy consists of sampling agents, which are grouped
according to their associated area. Each group has a leader. Thus, the second level of the hierarchy consists of sampler group leaders. Sampler group leaders are also grouped according to their associated area. Each group of sampler leaders is associated with a zone group leader. Thus, the third level of the hierarchy consists of zone group leaders, which in turn, are also grouped according to their associated area, with a zone group leader, and so on. Leader agents are responsible for retrieving and combining information from their group of agents. Members of a group are referred to as group subordinates. Sampling agents are mobile; therefore, they may change their group when moving to a different area. The sampler leaders are responsible for the movements of sampling agents.

A sampler agent takes measurements and forms capsules. These capsules are sent to the sampler leader at specified intervals. A sampler leader collects capsules from its sampler agents to represent its localInfo. In this computation, it uses the previous value of localInfo; it then sends its localInfo to its zone leader. A zone leader collects the localInfo of all the sub-leaders of its zone and forms a localInfo of its entire zone. In turn, it sends this localInfo to its leader and so on. The top zone leader, whose zone consists of the entire area, forms a localInfo of all the objects in the entire area. The algorithms for these agent processes are presented in the next section.
Figure 2: DDM hierarchy information flow diagram
Chapter 3

Architecture Algorithm Description

This chapter provides three algorithms suitable for gathering information from a very large number of agents. The first algorithm describes the data collection of raw inaccurate data; the second algorithm describes how each leader should integrate the collected data; and the third algorithm describes how each group leader should deploy agents to balance the information gathering load.

More specifically, the formation of a global information map integrates the following processes:

i. Each sampling agent gathers raw sensed data and generates capsules. Every dT seconds each sampler group leader obtains capsules from all its sampling agents and integrates them into its \( \textit{localInfo} \).

ii. Every dT seconds each zone group leader obtains \( \textit{localInfo} \) from each of its subordinate group leaders and integrates them into its own \( \textit{localInfo} \). As a result, the top-level group leader's \( \textit{localInfo} \) will contain a global information map.

iii. Every dT seconds each zone group leader asks its leader for agent redeployment instructions, calculates how many agents should move from one subzone to another and issues redeployment instructions for its subordinate leaders. The goal of this load balancing algorithm is to balance the ratio of agents per goal in every zone. Every dT seconds each zone group leader will use local information and directions from its leader to direct leaders to redeploy sampling agents below it.

In the presentation of the algorithms a dot notation is used to describe a field in a structure, e.g., if \( c = <t, sa, \{s_1, ..., s_i\} > \) then \( c.sa \) is the sampling agent field of capsule \( c \).
3.1. **Sampler capsule generation algorithm.**

One sampling agent is used to deduce a set of possible object states at a given time in the form of a capsule. A sampling agent takes $k$ consecutive measurements and creates a new capsule, $c$, such that the time of the capsule is the time of the last measurement. The state of the sampling agent while taking the measurements is assigned to $c.sa$. The object states are assigned to $c.states$. The object states' part is the result of activating the domain function $PosS$ on $k$ consecutive measurements. The agent stores the capsules until its sampler group leader asks for them. The group leader deletes the capsules after delivering them to the sampler agent.

3.2. **Leader localInfo generation algorithm**

Every $dT$ seconds each group leader performs the *localInfo* generation algorithm. Each group leader maintains its own *localInfo*. The leader first purges any data older than $\tau$ seconds before processing new data. Updating *localInfo* involves three steps: (i) obtaining new information from the leader’s subordinates; (ii) finding new paths; (iii) and merging the new paths into the *localInfo*.

In the first phase, every leader obtains information from its subordinates. The sampler group leader obtains information from all of its sampling agents from their *unusedCapsules* and adds them to its *unusedCapsules* set. The zone group leader obtains *localInfo* from its subordinates. It adds the *unusedCapsules* to its *unusedCapsules* and merges the *infoMap* of its subordinates’ *localInfo* to its own *localInfo*.

Merging of functions is performed both in steps (i) and (iii). Merging is needed since, as noted earlier, object state functions inserted by a leader into the information map are accepted by the system as correct and will not be removed. However, different agents may sense the same object and therefore it may be possible that different functions coming from different agents will refer to the same object. The agents should recognize such cases and keep only one of these functions in the *infoMap*. The following lemma is used to find identical functions and merge them.
Lemma 1:
Let $p^1 = \langle \pi^1_1, \ldots, \pi^1_z \rangle$, $p^2 = \langle \pi^2_1, \ldots, \pi^2_z \rangle$ be two paths, where $\pi^1_j = \langle t^1_j, sa^1_j, s^1_j \rangle$ and $\pi^2_j = \langle t^2_j, sa^2_j, s^2_j \rangle$ and $f^1_{\pi^1_1, \pi^1_2} = \text{pathToFunc}(p^1)$, $f^2_{\pi^2_1, \pi^2_2} = \text{pathToFunc}(p^2)$.

If $\text{ResBy}(\langle t^1_i, s^1_i \rangle, \langle t^2_i, s^2_i \rangle)$ then for any $f^1_{\pi^1_1, \pi^1_2}(t) = f^2_{\pi^2_1, \pi^2_2}(t)$

Since every two triplets in a given path follow the domain dependent $\text{ResBy}$ function, if two triplets from different paths follow the $\text{ResBy}$ function every two triplets from both pathes will follow the $\text{ResBy}$ function. Therefore, the path associated function of either path will result in the same for a given $t$.

Leaders use lemma (1) and the $\text{ResBy}$ relation to check whether the first state of an object state function resulted from the first state of a different object state function. If one of the states is related in such a way, the leader changes the minimum and the maximum triplets of the object state function. The minimum triplet is the starting triple that has the lowest time. The maximum triple is the ending triple that has the highest time. Intuitively, the two state functions are merged and the resulting function is associated with the combination of their ranges. If a leader cannot find an object state function to meet the subordinate’s function, the leader will add it as a new function to its $\text{infoMap}$.

The second step is performed by every leader and corresponds to finding paths and extending current paths given a set of capsules. In order to form paths from capsules, the agent should choose only one object state from each capsule. This constraint is based on the following lemma.

Lemma 2:
Let $C^1 = \langle t^1, sa^1, s^1_1, \ldots, s^1_y \rangle$, $C^2 = \langle t^2, sa^2, s^2_1, \ldots, s^2_y \rangle$ and $\text{ResBy}(\langle t^1, s^1_1 \rangle, \langle t^2, s^2_1 \rangle)$ and $\text{ResBy}(\langle t^1, s^1_i \rangle, \langle t^2, s^2_i \rangle)$ then

i. if $s^1_i \neq s^2_i$ then $s^1_j \neq s^2_j$

ii. if $s^1_i = s^2_i$ then $s^1_j \neq s^2_j$

iii. if $s^1_i = s^2_i$ then $s^1_j = s^2_j$
iv. if \( s_j^i = s_j^k \) then \( s_j^i = s_j^k \)

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. Such a case of two different states violates the \( ResBy \) constraints.

Every leader stores the correct object state functions as part of its \( infoMap \) structure. The top-level leader will also have represented object state functions with an intermediate probability to represent objects. The top leader knows that some of the paths that he would like to use to form state functions are correct but it cannot decide which are correct. Paths with only one viewpoint are paths that may be correct. For instance, in the ANTS domain, paths with one viewpoint will have a 50% probability of being correct, due to the characteristics of the sensors. In other domains, the characteristics of the sensors may lead to different probabilities. The top-level leader will use these paths of intermediate probability to form a set of functions that have a partial probability of being correct.
Chapter 4

Architecture Algorithm Complexity

The issues revolving around the complexity of the proposed hierarchical algorithm are addressed in this section. This includes the issue of whether a single level hierarchy or a multiple level one is better. If there is one level in the hierarchy then all the capsules are processed by the sampling leader agent. The advantage to this approach is within the simplicity of the system design. There is never a question as to which leader should process the capsules. Once two or more levels exist, several sampling agents become responsible for processing the capsules simultaneously. While this approach could save time, it also may result in a situation where all of the capsules are not able to be processed within the first iteration of the algorithm. This is because any capsules that could not be used in building state functions will find their way into the unusedCapsules set and will then be transferred to the zone leader. The zone leader will collect all of the unusedCapsules and will process them one more time. Thus, the second level may waste the time saved by the distribution in the first level. Therefore, the time benefit of the hierarchy depends on the ratio of the capsules that the lower level is able to use.

We will show that some balance exists between these two approaches. While adding levels within the hierarchy will aid in the system's speed, adding too many levels will cause the number of unused capsules to decrease the overall performance. In order to determine the proper balance between these extremes, we first analyze the complexity of the two main algorithms. Based on this result, we consider the algorithm for forming paths and then the algorithm that merges functions.

Lemma 3:
Let $C$ be a set of capsules and $m$ the maximum number of states in a capsule.
The time complexity of finding the paths by the algorithm of step 2 in the worse case is:
$$\frac{(|C| \cdot m)^2}{2}.$$
Lemma 4:
The time complexity of merging two sets of object state functions \( F^1 \) and \( F^2 \) in the worse case is: \( \frac{|F^1| \cdot |F^2|}{2} \).

The most time consuming process is the formation of new paths in step 2, which depends on the number of capsules generated by the agents. Thus we state this number in the next lemma.

Lemma 5:
Let \( O \) be the group of objects located in the area \( \Lambda \) in a given time period and \( A \) the set of agents located in the area. Let \( \lambda \) be the size of the sub-area sensed by a single sampling agent. Suppose that in a given \( \tau \) time period the sampling agent activates its sensor for \( \delta \) time periods. Let \( C \) be a set of capsules generated by agents in area \( \Lambda \) in the period \( \tau \). Then:

\[
|C| \leq \frac{\tau}{\delta} \cdot |O| \cdot |A|
\]

Intuitively, lemma 5 states that the number of capsules is bound by the number of objects that the agents may observe in a given time period. Using the above lemmas we derive bounds on the percentage of \textit{unusedCapsule} that should be processed at a given level to make it beneficial to add additional levels.

Theorem 1:
Let area \( \Lambda \) be divided into \( q \) subsections, \( \Lambda_i, 1 \leq i \leq q \) such that \( \Lambda = \bigcup_{i=q}^{1} \Lambda_i \) and \( \alpha \) be the capsule percentage that could not be used in the state function construction by the agents at a given level. Then, if \( \alpha < \frac{\sqrt{q^2 - 1}}{q^2} \) it is beneficial, with respect to performance time, to increase the hierarchy by one level, given that there are at least two agents in each area.

As can be seen in theorem 1 even when \( \alpha \) is very close to 1 it is still beneficial to consider adding an additional level.
Chapter 5

Load balance algorithm

The aim of the load balance algorithm is to wisely distribute the sensing agents when goals are not distributed evenly. First, we will show that when many goals are concentrated in a specific area the performances of a uniform distribution of agents may achieve poor results. Then we will show that in those cases, concentrating more agents in the area that contains more goals may improve the performances of the system. Moreover, in the long run the performances will be improved until the ratio between agents and goals will be equal among all parts of the system.

We developed a dynamic hierarchic load balancing algorithm to effectively distribute the sensing agents per goal ratio. Every dT seconds, each DDM group leading agent follows the same load-balancing algorithm. The algorithm uses the leader knowledge base to balance the ratio of sensing agents per goal between its subordinates and it is based on load balancing behavior in nature.

In nature we often see fields of potential that result in movement: Differences in the number of electrons between two places on a conductor may cause an electronic current [16]; In the atmosphere, high pressure and low pressure blocks of air may lead to wind blowing [72]; Differences in the temperature of water results in a current in the ocean [10, 28]. These phenomena are known for the greater importance of close interactions. These movements are found in nature to balance the potential fields.

We use the same principles that make electrons move from atoms with spare electrons to atoms with no electrons; air molecules move from high pressure to low pressure and water molecules move from the hot side of a box of water to the cold side. These principles are used to balance the agents per object, APO, ratio in the controlled zone of each zone group leader.

In the DDM each zone leader translates its knowledge base into a physics notation of Thermodynamics and Fluid Dynamics. It then applies the natural behavior of load balancing and finally it translates the physics solution back into the DDM notation. Taking advantage of
the greater importance of close interactions, DDM takes into consideration interactions only between nearest neighbors. This ensures a very low computation load. Following this solution the zone group leader directs its subordinate leaders on how many agents to transfer from one leader to another.

We demonstrate the load balancing mechanism to balance the agent per goal ratio between equal geographic zones. Such an example may result in a heavy computation load if the agents will concentrate above a certain degree in a few zones. This problem should be studied in the future and may be solved if the mechanism will balance unequal geographic zones. In that case the size of each zone should be dynamically decided.

To apply physics behavior we need to have a virtual location representation of zones, an in function and various distance measurements. The virtual location of zone \( z \) is denoted \( z.vl \).

The load balancing algorithm requires a domain dependent on in functions.

**Definition 6:**
The function \( \text{in}, \text{in}(agent.vl,zone.vl) \) receives agent and zone virtual states and returns a boolean answer to the question of whether the agent is located inside the zone.

The load balancing algorithm also requires three kinds of domain dependent distance functions.

**Definition 7:**
The domain dependent distance functions:

i. The first distance function, \( \text{distanceAA} \), receives two states of agents and returns a non-negative scalar. The \( \text{distanceAA} \) reflects the virtual distance between two agents. The \( \text{distanceAA} \) function is symmetric such that for agents \( agent \) and \( agent' \):

\[
\text{distanceZA}(agent.sa,agent'.sa) = \text{distanceZA}(agent'.sa,agent.sa)
\]

ii. The second distance function, \( \text{distanceZA} \), receives a virtual location representation of a zone and a state of an agent and returns a non-negative scalar. The \( \text{distanceZA} \) reflects the virtual distance between a zone and an
agent.

The distanceZA function is symmetric such that for agent agent and zone zone:

\[ \text{distanceZA}(\text{zone} \cdot \text{vl}, \text{agent} \cdot \text{sa}) = \text{distanceZA}(\text{agent} \cdot \text{sa}, \text{zone} \cdot \text{vl}) \]

iii. The third distance function, distanceZZ, receives virtual location representations of two zones and returns a non-negative scalar. The distanceZZ function reflects the virtual distance between two zones. The distanceZZ function is symmetric such that for zones zone and zone':

\[ \text{distanceZZ}(\text{zone} \cdot \text{vl}, \text{zone'} \cdot \text{vl}) = \text{distanceZZ}(\text{zone'} \cdot \text{vl}, \text{zone} \cdot \text{vl}) \]

<table>
<thead>
<tr>
<th>Physics</th>
<th>DDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential</td>
<td>APO</td>
</tr>
<tr>
<td>Potential field</td>
<td>Set of APO</td>
</tr>
<tr>
<td>Particle</td>
<td>Agent</td>
</tr>
<tr>
<td>Euclid's space</td>
<td>Zone virtual space</td>
</tr>
<tr>
<td>Distance</td>
<td>Virtual distance</td>
</tr>
<tr>
<td>Potential gradient</td>
<td>APO gradient</td>
</tr>
</tbody>
</table>

Figure 3: Translating the physics notation to DDM

Physics behavior dictates that particles will move according to the potential gradient from high potential to lower potential. Following this behavior, agents move according to the APO gradient from zones with high APO to zones with low APO.

**Definition 8**: The APO gradient is equal to the change of the APO divided by the change of the location, such that the APO gradient = \( \frac{dAPO}{dX} \) whereas dAPO is the difference of AOP values between two neighboring zones and dX is the virtual distance between the same two neighboring zones. APO gradient represents the imbalance between two neighboring zones.

Each zone group leader applies two algorithms: (i) redirection of agents and (ii) redeployment of agents. Sampler group leader applies only the latter.
The goal of the redirection of agents algorithm is to direct each subordinate leader as to (a) how many agents should leave it; and (b) to which subordinate leader it should refer each leaving agent. The leader that redirects an agent is denoted the dispatching leader while the referred leader is denoted the receiving leader. In general every dT seconds each zone group leader (i) calculates the APO ratio of all its subordinate zones; (ii) finds the two neighboring subzones with the greatest gradient; (iii) issues a redeployment order for one agent to move along the gradient; (iv) updates its knowledge base as if that agent already moved; and (v) repeats the process from step i until the greatest gradient is smaller than a predefined acceptable constant small gradient value, εG. The issued orders are kept until the subordinate leaders request them or they are deleted after a certain period of time.

The second algorithm, redeployment of agents algorithm, is responsible for processing the agent redeployment orders. According to this algorithm every dT seconds each group leader asks its leader for a set of orders. It receives from its leader only the stored orders that direct agents into or out of its zone. The group leader then follows order by order and executes it.

The following definitions are used to detail the algorithms:

**Definition 9**: Closest neighbor, CN, is a pair of zones, zone and zone’, such that distanceZZ(zone, vl, zone’, vl) ≤ εD

whereas εD is defined as the range of interaction between zones.

**Definition 10**: Closest neighbors of zone zone, CNs, is a set of zones such that

\[
CNs = \{zone_0...zone_{n_oz-1}\}
\]

where

i. \( n_oz \) is the number of zones in a CNs set;

ii. \( \forall zone_i, 0 \leq i < n_oz \)

\[
distanceZZ(zone, vl, zone_i, vl) \leq \varepsilon D
\]

**Definition 11**: A cell is a sequence such that cell_i = <na, no, Gs, vl> where

i. \( na \) is the number of agents in subzone i,
ii. $no$ is the number of known objects in subzone $i$.

iii. $Gs$ is the set of gradients between subzone $i$ and its closest neighbors.

iv. $vl$ is a virtual location representation of the subzone $i$.

A cell is a representation of the zone controlled by a subordinate zone group leader.

**Definition 12:** Cells is a set of cells such that $Cells = cell_0...cell_{nos-1}$ where $nos$ is the number of subordinates of the leader. Consequently, $cell_i$ represents the zone controlled by subordinate zone group leader $i$.

**Definition 13:** $\text{gradient}_{i,j}$ is the tuple $<i, j, value>$ where

i. $i$ is the index of the gradient origin cell

ii. $j$ is the index of the gradient ending cell

iii. $value = \frac{(cell_{i,na} - cell_{j,na})}{(cell_{i,no} - cell_{j,no})}$

iv. $distanceZZ(cell_{i,vl},cell_{j,vl})$ is a domain dependent function that describes the virtual distance between $cell_{i,vl}$ and $cell_{j,vl}$. $\text{gradient}_{i,j}$ represents the imbalance of $APO$ values between $cell_i$ and $cell_j$.

**Definition 14:** Gradients is a sub set of all possible gradients such that $Gradients \subseteq Cells \times Cells$ and

$\text{Gradient} \in \forall \text{gradient}_{i,j}, 0 \leq i, j < n, distanceZZ(cell_{i,vl},cell_{j,vl}) \leq \varepsilon D$

where $n$ is the number of subordinates of the leader.

**Definition 15:** An order is a sequence of $<i, j, dispatcher.vl, receiver.vl>$ where

i. $i$ is the index of the dispatching leader

ii. $j$ is the index of the receiving leader

iii. $dispatcher.vl$ is the virtual location of the dispatching leader zone
iv. \textit{receiver.vl} is the virtual location of the receiving leader zone

An \textit{order} represents a directive either to (i) a dispatching group leader \(i\) to dispatch one agent towards group leader \(j\) or to (ii) a receiving group leader \(j\) to receive one agent from dispatching group leader \(i\).

\textbf{Definition 16:} \textit{Orders} is a set of orders such that \(\text{Orders} = \text{order}_0 \ldots \text{order}_{\text{noo}-1}\) where \(\text{noo}\) is the number of orders in the set.

An orders set is used to aggregate orders for each \textit{dispatching leader} and for each \textit{receiving leader}.

\textbf{Definition 17:} \textit{OrdersByIndex} is a hashtable such that \textit{OrdersByIndex}: \textit{Cells} \rightarrow \textit{Orders}. \textit{Orders} sets are stored in this hashtable and the key for each set is an index of a \textit{cell} as it appears in \textit{Cells}.

This hashtable is used to store (i) deployment orders set by the \textit{dispatching leader} index; (ii) receiving \textit{Orders} set by the \textit{receiving leader} index.

\textbf{5.1. Redirection of agents algorithm}

The input for the \textit{redirection of agents} algorithm are (1) the target function set as stored in the information map of each leader, (2) the connection to the subordinate leaders and (3) a storage set of unprocessed orders to subordinates. The target state function is used to find the state of the detected objects. The connection to the subordinate leaders is needed to inquire about the number of their sensing agents. The information about the state of the detected object and the number of agents in each subzone is used to find the imbalance in each zone group leader's controlled zone. The last input of the algorithm, the storage of unprocessed orders to subordinates, is used to store the new orders for each subordinate
leader. Each order is stored until it is requested by the ordered subordinate leader or until a timeout is expired.

The redirection of agents algorithm is divided into three consecutive steps: (i) generating the Cells set; (ii) generating the Gradients set; and (iii) generating the OrdersToDispatchingSubordinates and OrdersToReceivingSubordinates hashtables.

In the first step of the redirection of agents algorithm each zone group leader forms a Cells set. During this step the leader assigns the number of objects and agents in each subordinate leader’s zone to the Cells set. The zone group leader uses the current time value to evaluate each state function in order to find the number of objects in each subordinate leader’s zone. For each resulted state it checks if it is virtually located within any of the subordinate leader’s zones. For this check it uses the domain dependent $in$ function. Each state that is found to be virtually located in subordinate leader’s $i$ zone increases the number of objects of $cell_i$ by one. As a result the number of objects field, $no$, of each cell will contain the number of detected objects that should be virtually located in the correlated subordinate leader’s zone.

To determine the number of agents in each subordinate leader’s zone, the zone group leader first processes information retrieved from its subordinate leaders and then uses orders that still have not been requested by the subordinate leaders. Even though the stored orders have not been executed, the zone group leader considers them as such. Since every order is stored both in the dispatching hashtable, OrdersToDispatchingSubordinates, and in the receiving hashtable, OrdersToReceivingSubordinates, the zone group leader considers only orders from one hashtable. The zone group leader reduces the number of agents that should be redeployed from the agent's count of the dispatching leader and adds it to the count of the receiving leader.

redirection of agents algorithm - (i) generating the Cells set

**Input:** the current time, $t$,
subordinate leaders $L$,
target function set $F$,
OrdersByIndex hashtable: OrdersToDispatchingSubordinates

**Output:** cells set, $Cell$
OrdersByIndex hashtable: OrdersToDispatchingSubordinates
// determine the number of objects in each subzone
for each subordinate leader i
{
    cell_i,sl = the virtual location properties of subordinate leader i's controlled zone
    for each state function, \( f^k \), in \( F \)
        if object state \( f^k(t) \) belongs to leader agent i's controlled zone
            cell_i,no++
}

// determine the number of agents in each subzone
// 1. process information from subordinate leader agents
for each subordinate leader agent i
    cell_i,na = ask subordinate leader agent i for its number of agents

// 2. process information from stored unprocessed orders
for each order set, orders, of OrdersToDispatchingSubordinates
    for each order in orders
    {
        i = order.i
        j = order.j
        cell_i,na--
        cell_j,na++
    }

The second step of the redirection of agents algorithm is responsible for calculating the imbalance between each two close neighboring subordinate leaders’ zones. According to this step every combination of two cells is checked to see if they are close neighbors. The test to determine if they are close neighbors is if the domain dependent distanceZZ function returns a value that is smaller than a predefined constant \( cD \) for the two questionable cells.

For every pair of close neighbors the zone group leader finds the value of the gradient of APO. The value of the gradient is the difference between the APO values of the cells divided by the virtual distance between them. It represents the imbalance between the cells and its direction. The zone group leader then adds the gradients to the Gradients set. It also adds the gradient to the set of gradients of the first cell in the neighboring pair. The first cell represents the origin of the gradient. The same gradient with an opposite sign will be added to the second cell of the pair when the second cell will become the origin. At the end of this step the Gradients set is sorted according to the gradient value, from greater to the lower. In this stage
every cell contains the gradients of all its close neighbors, the number of agents and the number of objects in its correlated subordinate leader’s zone.

**redirection of agents algorithm - (ii) generating the Gradients set**

**Input:** cells set, \( \text{Cell} \),

**Output:** gradients set, \( \text{Gradients} \)

for each \( \text{cell}_i \) in Cells

\[
\begin{align*}
\text{for each } \text{cell}_j \text{ in Cells and } \text{cell}_i & \neq \text{cell}_j \\
\{ & \\
\text{if } \text{dis tan } \text{ce}(\text{cell}_i, \text{cell}_j) \leq \varepsilon D \\
\{ & \\
\text{gradient}_{i,j},i = i \\
\text{gradient}_{i,j},j = j \\
\text{gradient}_{i,j}.\text{value} = & \frac{\left( \begin{array}{c} \text{cell}_i,na - \text{cell}_j,na \\ \text{cell}_i,\text{no} - \text{cell}_j,\text{no} \end{array} \right)}{\text{dis tan } \text{ce}(\text{cell}_i, \text{cell}_j)} \\
\text{Gradients} = & \text{Gradients} \cup \text{gradient}_{i,j} \\
\text{cell}_i,\text{gradients} = & \text{cell}_i,\text{gradients} \cup \text{gradient}_{i,j} \\
\} & \\
\} & \\
\} & \\
\} & \\
\} & \\
\} & \\
\}
\end{align*}
\]

sort \( \text{Gradients} \) according to the gradient value from greater to the smaller.

In the last step of the *redirection of agents* algorithm, the zone group leader issues orders to its subordinate leaders. In this step the leader uses the sorted \( \text{Gradients} \) set to determine how many orders to issue. The leader keeps the \( \text{Gradients} \) set sorted for that purpose. According to this step, the leader follows a loop as long as the value of the first element of the \( \text{Gradients} \) set is greater than a predefined constant \( \varepsilon G \). This ensures that the gradients value between every two neighboring cells is less than or equal to \( \varepsilon G \). That constant represents the system's tolerance degree to the imbalance. In every cycle of the loop the leader issues an order to move one agent from one side of the greatest gradient to the other side.

The leader holds two hashtables of orders: \( \text{OrdersToDispatchingSubordinates} \) and \( \text{OrdersToReceivingSubordinates} \). \( \text{OrdersToDispatchingSubordinates} \) stores orders sets for
the dispatching subordinate leaders. The hashtable key represents the dispatching leader's index as it appears in the subordinate leaders $L$ set. $OrdersToReceivingSubordinates$ stores orders sets for the receiving subordinate leaders. The hashtable key represents the receiving leader's index as it appears in the subordinate leaders $L$ set.

When a new order is issued the leader updates the $OrdersToDispatchingSubordinates$ hashtable in the following manner:

(i) if there were no previous orders for the dispatching subordinate leader it creates a new Orders set and adds the new order to it;

(ii) if there was already a set of orders it adds the new order to it.

The leader follows the same procedure for the receiving subordinate leader to update the $OrdersToReceivingSubordinates$ hashtable.

After issuing a new order and updating both $OrdersToDispatchingSubordinates$ and $OrdersToReceivingSubordinates$ hashtables, the zone group leader updates the number of agents in the dispatching and receiving cells. It increases the number of agents in the receiving cell by one and decreases the number of agents in the dispatching cell by one. It then updates all the gradients associated with the dispatching and the receiving cells and relocates the updated gradients in the sorted $Gradients$ set to keep it sorted. This final step of the redirection of agents algorithm ends when the leader is satisfied with the predicted results of its orders.

**redirection of agents algorithm - (iii) generating the $OrdersToDispatchingSubordinates$ and $OrdersToReceivingSubordinates$ sets**

**Input:** gradients set, $Gradients$,
cells set, $Cell$,
$OrdersByIndex$ hashtable: $OrdersToDispatchingSubordinates$,
$OrdersByIndex$ hashtable: $OrdersToReceivingSubordinates$

**Output:** updated $OrdersByIndex$ hashtable: $OrdersToDispatchingSubordinates$,
updated $OrdersByIndex$ hashtable: $OrdersToReceivingSubordinates$

$maxGradient = \text{the first element of } Gradients$

while (maxGradient.value $\geq \varepsilon G$)

{
  i = maxGradient.i
  j = maxGradient.j
  // issue a new order for the dispatching subordinate leader
  if $OrdersToDispatchingSubordinates_i$.exists( i )
    orders = $OrdersToDispatchingSubordinates_i$.get( i )

}
else
    order = new empty Orders set
orders = orders ∪ < i, j, cell_i, vl, cell_j, vl >
OrdersToDispatchingSubordinates_j.put(orders)
// issue a new order for the receiving subordinate leader
if OrdersToReceivingSubordinates_j.exists( i )
    orders = OrdersToReceivingSubordinates_j.get( i )
else
    order = new empty Orders set
orders = orders ∪ < i, j, cell_i, vl, cell_j, vl >
OrdersToReceivingSubordinates_j.put( orders )
// update knowledge base
  cell_i, na --
  cell_j, na ++
// update gradients influenced by the reduction of an agent from cell_i
for each gradient g in cell_i, gradients
{
    \[
    \frac{(cell_{g,j}, na - cell_{g,j}, no)}{(cell_{g,i}, no - cell_{g,j}, no)}
    \]
    dis tan ce(cell_{g,i}, vl, cell_{g,j}, vl)
    \]
    revaluate g.value:
    g.value = \[
    \frac{(cell_{g,j}, na - cell_{g,j}, no)}{(cell_{g,i}, no - cell_{g,j}, no)}
    \]
    dis tan ce(cell_{g,i}, vl, cell_{g,j}, vl)
    \]
    relocate g in the ordered Gradients set
}
// update gradients influenced by the increment of an agent of cell_j
for each gradient g in cell_j, gradients
{
    \[
    \frac{(cell_{g,i}, na - cell_{g,j}, na)}{(cell_{g,i}, no - cell_{g,j}, no)}
    \]
    dis tan ce(cell_{g,i}, vl, cell_{g,j}, vl)
    \]
    revaluate g.value:
    g.value = \[
    \frac{(cell_{g,i}, na - cell_{g,j}, na)}{(cell_{g,i}, no - cell_{g,j}, no)}
    \]
    dis tan ce(cell_{g,i}, vl, cell_{g,j}, vl)
    \]
    relocate g in the ordered Gradients set
}
maxGradient = the first element of Gradients

5.2. Redeployment of agents algorithm

The aim of the redeployment of agents algorithm is to process the orders received from its leader. Every dT seconds each leader follows the redeployment of agents algorithm.
This algorithm directs the leader to either (i) send agents to a zone that is located outside of its own boundaries; or (ii) receive agents from a zone that is located outside of its boundaries. Since zone group leaders are not directly responsible for sampling agents, they delegate subordinate leaders to process orders. Sampling group leaders are the only leaders that may directly order agents to change their location. Therefore, the redeployment of agents algorithm is implemented differently by zone group leaders and by sampling group leaders.

**Zone group leader implementation for the redeployment of agents algorithm**

The goal zone group leaders' redeployment of agents algorithm is to find the right subordinate leader to process each order. The redeployment of agents algorithm is divided into two parts: (i) the first, the dispatching orders algorithm, is responsible for processing orders to send agents to other zones; (ii) the second, the receiving orders algorithms, is responsible for processing orders to prepare to receive agents from other zones. Both parts make use of a function that finds the closest cell index to a given cell's virtual location.

The **Find the closest cell index to a cell virtual location** function receives a set of cells and a virtual location of a cell. It uses the domain dependent distanceZZ function that receives two cells' virtual locations and returns a non-negative scalar. The **Find the closest cell index to a cell's virtual location** function uses the distanceZZ function to compare all the distances between the given cells' locations and any cell element of the set. The index of the element in the cells set that resulted in the greatest distance for the given cell is returned. The third parameter, agents-needed, is a boolean flag parameter that indicates whether to compare only distances from cells that contain agents. Only cells with agents are required when dispatching orders are processed; whereas this rule is not needed when receiving orders are processed.

```
Find the closest cell index to a cell's virtual location

**Input:** cells set, Cells
           cell virtual location, cell-vl
           boolean value, agents-needed

**Output:** cell index

closest-cell-index = -1  // return value of –1 means that no suitable cell is found
for each cell, i in Cells
{
   if ! agents-needed || cell,na > 0
   
```

40
if \( i = 0 \)
{
   closest-cell-index = i
   closest-distance = \text{distanceZZ}(cell_i, vl, cell-vl)
}
else if \text{distanceZZ}(cell_i, vl, cell-vl) < closest-distance
{
   closest-cell-index = i
   closest-distance = \text{distanceZZ}(cell_i, vl, cell-vl)
}
}
}
return closest-cell-index

The \textit{dispatching orders} part of the \textit{redeployment of agents} algorithm delegates dispatching orders to subordinate leaders and updates the \textit{OrdersToDispatchingSubordinates} hashtable. It receives (1) a set of cells, (2) a communication link to its zone group leaders and (3) a hashtable with orders to its subordinates to dispatch agents. The leader first approaches its zone group leader and asks for orders to dispatch agents from its zone to an external zone. It then executes order by order and finds the closest subcell index, \( i \), that still contains agents. If subordinate group leader \( i \) already has orders in the \textit{OrdersToDispatchingSubordinates} hashtable, the leader adds a new order to its set of orders. The order is a dispatching order for the found cell. It specifies that the subordinate group leader \( i \) should send an agent to the cell denoted by the processed order. If the leader does not have previous orders to dispatch agents, it creates a new empty set of orders, adds the new order to it and puts it in the \textit{OrdersToDispatchingSubordinates} hashtable. For any new dispatching order the leader updates the knowledge base by decreasing the number of agents in cell \( i \) by one. In this manner the leader updates the hashtable of dispatching orders and the set of cells.

**Dispatching orders**

\textbf{Input:} cells set, \textit{Cells}  
Zone group leader \( l \),  
\textit{OrdersByIndex} hashtable: \textit{OrdersToDispatchingSubordinates}  

\textbf{Output:} updated cells set, \textit{Cells}  
updated \textit{OrdersByIndex} hashtable: \textit{OrdersToDispatchingSubordinates}  
orders-from-leader = ask leader, \( l \), for orders as a dispatching subordinate leader
for each order, o, in orders-from-leader
{
   // get the closest cell index to the receiving cell
   i = closest-cell (Cells, o.receiving-vl, true)
   if i = -1 break
   if OrdersToDispatchingSubordinates[i].exists(i )
       orders = OrdersToDispatchingSubordinates[i].get(i )
   else
       order = new empty Orders set
   orders = orders ∪ < i, o.j, cell, .vl, o.receiving − vl >
   OrdersToDispatchingSubordinates[i].put( orders )

   // update knowledge base
   cell, na --
}

The receiving orders part, like the previous part, receives (1) a set of cells, (2) a set of leaders and (3) a hashtable. In this case the hashtable contains orders for subordinate leaders to prepare for incoming agents. The goal of this part is to delegate receiving orders to subordinate leaders. It requests its receiving orders from its zone group leaders. It then follows order by order and finds the closest cell index, i. The closest cell may not have any agents since the receiving orders do not require any preexisting agents. It checks whether the subordinate leader i has previous orders in OrdersToReceivingSubordinates hashtable. If subordinate leader i does not have such orders, it creates an empty set of orders, adds a new order and puts the new set in the hashtable. If it has previous orders, the leader adds a new order to the existing set. The new order directs subordinate leader i to receive an agent from a zone that is specified in the processed order. Each new generated order is followed by updating the knowledge base. In this situation the number of agents in cell i will be increased by one.

**Receiving orders**

**Input:** cells set, Cells
Zone group leader l,
OrdersByIndex hashtable: OrdersToReceivingSubordinates

**Output:** updated cells set, Cells
updated OrdersByIndex hashtable: OrdersToReceivingSubordinates
orders-from-leader = ask leader, l, for orders as a receiving subordinate leader
for each order, o, in orders-from-leader
{
    j = closest-cell (Cells, o.dispatcher-vl, false)
    if i = -1 break
    if OrdersToReceivingSubordinates_j.exists( i )
        orders = OrdersToReceivingSubordinates_j.get( i )
    else
        order = new empty Orders set
        orders = orders ∪ < o.i, j, o.dispatcher = vl, cell, vl >
    OrdersToReceivingSubordinates_j.put( orders )
    // update knowledge base
    cell, na ++
}

**Sampling group leader implementation for the redeployment of agents algorithm**

In contrast to the zone group leader, the sampling group leader is directly responsible for the sampling agents. The aim of the redeployment algorithm in this case is (i) to find the right sampling agent and send it to the appropriate zone for dispatching orders or (ii) to mark the number of incoming new agents for receiving orders.

The *dispatching orders* part of the *redeployment of agents* algorithm receives a set of agents and a communication link to the zone group leader. The agents’ set represents the agents currently controlled by the leader. In the *dispatching orders* part of the algorithm the sampling group leader asks for all the dispatching orders from its zone group leader. It then follows order by order and chooses the agent that is closest to the zone of destination. After it finds the closest agent, the leader sends it a message to change its location towards the zone of destination. The agent will change its state according to the problem domain. To find the closest agent to a given zone, the sampling agent makes use of a function termed *find the closest agent index to a cell virtual location* function. This function receives a set of agents and a virtual location of a cell. It returns an index of an agent in the agents set. The index is the index of the closest agent to the given cell.

The *find the closest agent index to a cell virtual location* function uses the domain dependent *distance* function to find the virtual distance between an agent state and a virtual location of a cell. The leader compares the distance between every agent and the cell and returns the index of the agent that resulted in the shortest distance.
**Find the closest agent index to a cell virtual location**

**Input:** agents set, \( A \)
- cell virtual location, \( cell-vl \)

**Output:** agent index

\[ closest-agent-index = -1 \]  // return value of –1 means that no agent is found

for each agent, \( a_i \), in \( A \)

\{  
  if \( i = 0 \)
  \{  
    closest-agent-index = i  
    closest-distance = distanceZA(cell-vl, a_i.vl)  
  \}

  if \( distanceZA(cell-vl, a_i.vl) < closest-distance \)
  \{  
    closest-agent-index = i  
    closest-distance = distanceZA(cell-vl, a_i.vl)  
  \}
\}

return closest-agent-index

---

**Dispatching orders**

**Input:** agents set, \( A \)
- Zone group leader, \( l \)

**Output:** updated agents set, \( A \)

orders-from-leader = ask leader for orders as a dispatching subordinate leader

for each order, \( o \), in orders-from-leader

\{  
  a = Find the closest agent index to a  
  cell virtual location function (A, o.receiving-vl)  
  send agent \( a \) to o.receiving-vl  
  remove agent \( a \) from \( A \)  
\}

---

The *receiving orders* part of the *redeployment of agents* algorithm receives a communication link to the zone group leader and the number of agents that are on their way to the leader’s zone. The *receiving orders* part returns an updated number of incoming agents. It starts by requesting the receiving orders from the zone group leader. It then counts the number of orders and adds this figure to the amount of agents that are on their way to the zone.
**Receiving orders**

**Input:** Zone group leader, \( I \)
- number-of-incoming-agents

**Output:** updated number-of-incoming-agents

\[
\text{orders-from-leader} = \text{ask leader for orders as a receiving subordinate leader} \\
\text{number-of-incoming-agents} = \text{number-of-incoming-agents} + \text{length(orders-from-leader)}
\]

**Joining agents**

Finally, when an agent enters a zone of a sampling group leader it sends the leader a message declaring its arrival. Consequently, the sampling group leader executes the joining agent algorithm. This algorithm receives a communication link to the incoming agent, the set of agents that belong to the sampling zone and the number of agents that are currently on their way to the zone. It adds the joining agent to the agents set. It then sends the sampling agent a message that contains the properties of the zone and directions of how to change states while remaining under its control. The leader concludes by decreasing the number of incoming sampling agents by one.

**Joining agent**

**Input:** communication link to an incoming agent, \( a \)
- agents set, \( A \)
- number-of-incoming-agent

**Output:** updated agents set, \( A \)
- updated number-of-incoming-agents

Add \( a \) to the \( A \) set
apply change of states roles on the incoming agents

\[
\text{number-of-incoming-agents} = \text{number-of-incoming-agents} - 1
\]
Chapter 6

Load balance complexity

This section describes the time complexity of the load balance mechanism. The load balance algorithm comprises three parts: (i) the Redirection of agents algorithm; (ii) the Redeployment of agents algorithm and (iii) the sampling group leader's behavior when a new agent joins its controlled zone. While DDM zone group leaders and sampling group leaders apply both the first and the second algorithms, only the sampling group leaders implement the latter.

6.1. Redirection of agents algorithm

The Redirection of agents algorithm is implemented in the same manner by both zone group leaders and sampling group leaders. To present the time complexity of the algorithm first we will detail the time complexity of all of its parts. The Redirection of agents algorithm comprises a sequence of three phases:

i. Generating the Cells set

ii. Generating the Gradients set

iii. Generating the OrdersToDispatchingSubordinates and OrdersToReceivingSubordinates sets

Lemma 6:
The time complexity of Generating the Cells set phase is in the worse case: $|O| + |A|$. 

Proof:
In the first part of Generating the Cells set phase each state function is checked against each subordinate. Each state function represents a tracked object and the number of subordinates, $nos$, is constant. Therefore, the order of the first part is $nos$ multiplied by the
number of objects, \( no \). Since the number of subordinates is determined once and stays constant, the time complexity of this part is \( no \). Given that \( no \) is bound by \(|O|\) the time complexity is \(|O|\). In the second part of \textit{Generating the Cells set} phase, the leader asks each of its subordinate leaders for their number of agents. The time complexity of this part is therefore \( nos \). The last part of \textit{Generating the Cells set} phase, the leader runs through all of the unprocessed orders. The number of orders depends on the number of agents that should move from one subzone to another. The number of agents is denoted by \( na \) and it is bound by the number of agents in the system, \(|A|\). Combining all parts of \textit{Generating the Cells set} phase shows that the time complexity in the worse case is \(|O| + |A|\).

**Lemma 7:**
The time complexity of \textit{Generating the Gradients set} phase is \( O(1) \).

**Proof:**
In first part of \textit{Generating the Gradients set} phase, the leader checks any combination of two subcells to find neighboring cells. It forms a gradient for each of these pairs. The time complexity of such actions is \( O(1) \) since the number of subcells is constant. In the second part the leader sorts the set of gradients. Since the size of the set is constant, the time complexity of the second part is also \( O(1) \). Combining both parts of \textit{Generating the Gradients set} phase leads to the conclusion that the time complexity of the entire \textit{Generating the Gradients set} phase is \( O(1) \).

**Lemma 8:**
The time complexity of \textit{Generating the OrdersToDispatchingSubordinates} and \textit{OrdersToReceivingSubordinates} sets phase is \( |A| \).

**Proof:**
In this phase the leader balances the \( APO \) ratio between all subcells. It virtually moves agents from a high \( APO \) cell to a low \( APO \) cell. The leader virtually moves one agent at a time. In the worse case, all agents are located in \( cell_i \), all the objects are in \( cell_j \) and
distanceZZ(zone, vl, zone, vl) results in the longest distance between any subcells. In the worse case that distance passes through all the cells. In this case all the agents should move from one place while visiting all cells.

Since every loop iteration virtually moves one agent from one cell to its neighbor, the leader will use na iterations to virtually move all its agents from one cell to its neighbor. To move one agent from one place to another through all the cells, the leader will have to use nos-1 iterations. Therefore, in order to move all the agents from the origin cell to the goal cell through all the cells the leader will have to use na * (nos - 1) iterations.

In any loop iteration the leader revaluates all the gradients between the dispatching cell and its closest neighbors and between the receiving cell and its closest neighbors. It relocates each of these gradients in its sorted set. Since the number of cells is constant, the time complexity of each iteration is O(1). Given that the number of subordinate agents is constant, and that na is bounded by |A|, the time complexity of Generating the OrdersToDispatchingSubordinates and OrdersToReceivingSubordinates sets phase is |A|.

**Theorem 2:**
The time complexity of the redirection of agents algorithm is in the worse case |O| + |A|.

**Proof:**

Given that the time complexities in the worse case of redirection of agents algorithm phases are:

i. Generating the Cells set: |O| + |A|

ii. Generating the Gradients set: O(1)

iii. Generating the OrdersToDispatchingSubordinates and OrdersToReceivingSubordinates sets: |A|

The time complexity of the entire redirection of agents algorithm is in the worse case |O| + |A|. 
6.2. **Redeployment of agents algorithm**

In contrast to the *Redirection of agents algorithm* the *Redeployment of agents algorithm* is implemented differently by zone group leaders and by sampler group leaders. Therefore, the evaluation of the time complexity is divided into two different sections: the zone group leader implementation and the sampler group leader implementation.

**Complexity of implementation of the zone group leader's Redeployment of agents algorithm**

The zone group leader's implementation of the *Redeployment of agents algorithm* is comprised of two parts: (i) the *Dispatching orders* part and (ii) the *Receiving orders* part. In the first part the leader processes the incoming dispatching orders while in the second part it processes the receiving orders. Both parts use a function that finds the subcell closest to a given subcell, therefore, we will begin by evaluating the time complexity of this function.

**Lemma 9:**
The time complexity of Find the closest cell index to a cell virtual location function is $O(1)$.

**Proof:**
In this function, the zone leader follows cell by cell in a loop. It keeps the cell that results in the shortest distance to a virtual location of a given cell. Since the number of cells is constant, the time complexity of this loop is $O(1)$.

**Lemma 10:**
The time complexity of the *Dispatching orders* part is in the worse case $|A|$.

**Proof:**
According to the *Dispatching orders* phase, the zone leader asks its leader for a set of dispatching orders. It then follows each order in a loop and puts it in the $OrdersToDispatchingSubordinates$ hashtable. The leader activates *Find the closest cell index*
to a cell virtual location function for each order. In the worse case the leader is ordered to
move all the agents. Therefore, in the worse case there will be $|A|$ iterations of the loop. The
time complexity in the worse case will then be $|A| \cdot O(1)$ and therefore $|A|$.

**Lemma 11:**
The time complexity of the *Receiving orders* part is in the worse case $|A|$.

**Proof:**
In the *Receiving orders* phase, as in the *Dispatching orders* phase, the zone leader
asks its leader for a set of orders. In this phase these orders are receiving orders. It follows
each order in a loop and puts it in the $OrdersToReceivingSubordinates$ hashtable. In this
phase the leader also activates *Find the closest cell index to a cell virtual location* function
for each order. In the worse case the leader is ordered to receive all the agents. Therefore, in
the worse case there will be $|A|$ iterations of the loop and the time complexity in the worse
case will also be $|A|$.

**Theorem 3:**
The time complexity of the zone group leader's implementation of the *Redeployment of
agents algorithm* is in the worse case $|A|$.

**Proof:**
Given that the time complexity in the worse case of the zone group leader's
implementation of *Redeployment of agents algorithm* ingredients is:

i. Find the closest cell index to a cell virtual location: $O(1)$

ii. *Dispatching orders*: $|A|$

iii. *Receiving orders*: $|A|$

The time complexity of the entire zone group leaders' implementation of *Redeployment
of agents algorithm* in the worse case is $|A|$.
Complexity of the sampling group leader's implementation of the Redeployment of agents algorithm

As in the zone group leader implementation, the sampling group leader's implementation is comprised of two parts: (i) the Dispatching orders part and (ii) the Receiving orders part. The sampling group leader also has a function that finds the closest agent to a given subzone. In the sampling group leader's implementation, only the Dispatching orders part uses this function.

Lemma 12:
The time complexity of Find the closest agent index to a cell virtual location function in the worse case is $|A|$.

Proof:
In this function the sampling leader follows agent by agent in a loop. It keeps the agent that results in the shortest distance of a virtual location to a given cell. In the worse case all the agents are concentrated under the sampling leader. In this case there will be $|A|$ iterations of the loop.

Lemma 13:
The time complexity of the Dispatching orders part is in the worse case $|A|^2$.

Proof:
According to the Dispatching orders phase, the sampling leader asks its leader for a set of dispatching orders. It then uses Find the closest agent index to a cell virtual location function to find an agent and sends it to its receiving cell. In the worse case the number of dispatching orders may reach the number of agents $|A|$. Since the time complexity of Find the closest agent index to a cell virtual location function is $|A|$, the time complexity of activating this function $|A|$ times is $|A|^2$. 
Lemma 14:
The time complexity of the *Receiving orders* part is $O(1)$.

Proof:
In the *Receiving orders* part, the sampling leader asks its leader for a set of orders to receive agents. It finds the number of new orders and adds it to its counter. Since the time complexity of finding the length of a set is $O(1)$ the time complexity of all the *Receiving orders* part is $O(1)$.

Theorem 4:
The time complexity of the sampling group leader's implementation of the *Redeployment of agents algorithm* is in the worse case $|A|^2$.

Proof:
Given that the time complexity in the worse case of the zone group leader's implementation of *Redeployment of agents algorithm* ingredients is:

1. Find the agent index closest to a cell's virtual location: $|A|$
2. Dispatching orders: $|A|^2$
3. Receiving orders: $|A|^2$

The time complexity of the entire sampling group leader's implementation of the *Redeployment of agents algorithm* is in the worse case $|A|^2$.

Jointing agent
Lemma 16:
The time complexity of the *Joining agent* function is $O(1)$.

Proof:
According to *Joining agent*, the sampling leader communicates with an incoming agent and informs it about its change of states rules. This information exchange occurs once upon an agent's entry into the leader's controlled zone.
Chapter 7

The ANTS domain example: An implementation of DDM

The Autonomous Negotiating Teams (ANTS) program is a program sponsored by the Defense Advanced Research Projects Agency. The aim of the ANTs program is to seek novel approaches to deal with challenging distributed resource allocation problems. The key tasks of ANTs' challenge are to detect and track moving objects while using low cost hardware. A Doppler radar was chosen as the low cost hardware. Each of the radar sensors can generate electromagnetic pulses to measure nearby objects. Every sensor has three 120 degrees sectors while only one sector may be active at a time to detect objects. Each of these sensors may move and spin around its center. In the ANTs domain a sampling agent state is represented by the location of the sensor and its orientation.

In this chapter we demonstrate how to apply the DDM architecture and the load balance mechanism on a specific domain. We begin with the implementation of the sampler generation algorithm, and then show how to implement the domain dependent functions: the $\text{ResBy}$ function and the $\text{PosS}$ function. We conclude with a demonstration of the implementation of the information map generation process for ANTs' challenge.

In the following sections we will use the following data structures:

**Target state:** $s = \langle \hat{D}, \hat{V} \rangle$ where $\hat{D}$ is the location of the target and $\hat{V}$ is its velocity. For example: $\langle\langle100,100\rangle, <2,-1\rangle\rangle$.

**Sensing agent state:** $sa = \langle \hat{D}, O \rangle$ where $\hat{D}$ is the location of the sensor and $O$ is the orientation of the sensor. For example: $\langle\langle150, 150\rangle, \pi/2\rangle$.

**Capsule:** $c = \langle t, sa, \{s_1, s_2\} \rangle$ where $t$ is the time of the sampling, $sa$ is the sensing agent state and $s_1, s_2$ are two possible target states as found by the sampler in $sa$. For example: $\langle30, \langle\langle150, 150\rangle, \pi/2\rangle, \langle\langle100, 100\rangle, <2,-1\rangle\rangle, \langle\langle200, 200\rangle, <2,-3\rangle\rangle \rangle$.

**Path point:** $\pi_i = \langle t_i, sa_i, s_j \rangle$ where $t_i$ is the time of the $i$ point, $sa_i$ is the sensing agent's state of $i$ point and $s_j$ is the target state of $i$ point. For example: $\langle30, \langle\langle150, 150\rangle, \pi/2\rangle, \langle\langle100, 100\rangle, <2,-1\rangle\rangle\rangle$.
**Path:** \( p = \langle \pi_1, \pi_n \rangle \) where \( \pi_1 \) and \( \pi_n \) are the first and the last path points. Every two path points in a path follow the ResBy relation.

**Target state function:** \( f_{\pi_i, \pi_e}(t) = \langle \pi_s, V, \pi_s, V \cdot (t - \pi_s, t), \pi_s, V \rangle \) valid in the range of \( \pi_s, \pi_e \).

For example: if
\[ \pi_s = < 30, << 150, 150 >, \pi/2 >, << 100, 100 >, < 2, -1 > > > \text{ and } \]
\[ \pi_e = < 40, << 450, 25 >, \pi/4 >, << 120, 90 >, < 2, -1 > > > \text{ then } \]
\[ f_{\pi_i, \pi_e}(t) = << 100, 100 > + < 2, -1 > \cdot (t - 30), < 2, -1 > > > . \]

We may see that in \( t = 20 \) we will attain \( f_{\pi_i, \pi_e}(30) = << 100, 100 >, < 2, -1 > > > \) and in \( t = 40 \) we will obtain \( f_{\pi_i, \pi_e}(40) = << 120, 90 >, < 2, -1 > > > \).

For simplicity sake, we will refer to the function as \( f(t) = \langle D(t), V \rangle \) and to its properties as: \( f(D(t)), f(V(t)), f(t_s) \) and \( f(t_e) \).

**Local information map:** \( << f^1, ..., f^h >, < p_i, ..., p_l >, < c_i, ..., c_m > > > \)

Agents should turn raw sensed data into an information map. To form the local information map from raw sensed data, agents should adhere to the following five steps of data evolution: (i) measurements, (ii) capsule, (iii) path, (iv) target state function and (v) local information map. Sensing agents will be responsible for the measurements and for the formation of capsules. Leaders at all levels will use the generated capsules and former data to form new paths, target state functions and local information maps.

### 7.1. Sampler capsule generation algorithm.

We use one sampling agent's measurements (step 1) to deduce a set of possible target states at a given time in the form of a capsule (step 2). A sampling agent takes four consecutive measurements. Then it creates a new capsule, \( c \), such that the time of the capsule is the time of the last measurement. The state of the sampling agent while taking the measurements is assigned to \( c.sa \). The target states resulting from the application of the function \( rawDataTransformation \) to the four consecutive measurements is assigned to...
c.states. The agent stores the capsules until its leader asks for them. At that time, it sends the capsules to its sampler group leader. After delivering the capsules to the group leader the sampler agent deletes them. We will now demonstrate how an agent transforms four consecutive measurements into a capsule.

A measurement is a pair of amplitude, $\eta$, 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 of a Doppler radar the target is located based on the following equation:

\begin{equation}
R_i^2 = \frac{k \cdot e^{-\frac{(\theta_i - \beta)^2}{\sigma}}}{\eta_i}
\end{equation}

where, for each sensed target, $i$, $R_i$ is the distance between the sensor and $i$; $\theta_i$ is the angle between the sensor and $i$; $\eta_i$ is the measured amplitude of $i$; $\beta$ is the sensor beam angle; and $k$ and $\sigma$ 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 [6, 8]. Using the intersection method is very problematic in large scale systems as it requires full synchronization and cooperation between groups of three Dopplers. Thus, DDM uses measurements from only one Doppler to deduce possible target states.

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

\begin{equation}
\bar{D}_1 = \bar{D}_0 + \int_{t_0}^{t_1} \bar{V} dt
\end{equation}

where $\bar{D}_t$ is the displacement of the object at time $t$. If we consider the distance from the center of the Doppler we can claim that

\begin{equation}
R_t = R_0 + \int_{t_0}^{t_1} v_r dt
\end{equation}

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 in a short time period is zero. The next target location after a very short period of time is then:

\begin{equation}
R_i = R_0 + V_z \cdot (t_i - t_0)
\end{equation}

We denote \((t_i - t_0)\) by \(t_{i,0}\). Knowing the relation between \(R, \theta\) and \(\eta\), as presented in (1), we can find the next angle as a function of the former.

In Figure 4 the dark arrow represents a target movement vector and the circles along the target movement represent target locations, \((R_0, \theta_0)\), \((R_1, \theta_1)\) and \((R_2, \theta_2)\), at time \(t_0\), \(t_1\) and \(t_2\) as sensed by the Doppler. Following the projection of \(R_1\) and \(R_2\) over \(R_0\), \(R_{1,R_0}\) and \(R_{2,R_0}\) respectively, as presented by the dotted line, we reach the trigonometric equation:

\begin{equation}
\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)}
\end{equation}

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

Following equation 1, \(\theta_i\) can be described as

\begin{equation}
\theta_i = \beta \pm \sqrt{-\sigma \cdot \ln \left( \frac{\eta}{k} \cdot \frac{R_i^2}{R^2} \right)}
\end{equation}
By assigning $R_i$ as represented in (4) to equation (6) we can now deduce that the location, $(R_i, \theta_i)$, at $t_i$ of a sensed target may be written as a function of $\theta_0$ as follows:

**Theorem 5:** Assuming that the acceleration of a target in a short time period is zero, the next location of the target after a very short time, $t_{i,0}$, is then obtained by

\[
\theta_i(\theta_0) = \beta \pm \sqrt{-\sigma \cdot \ln \left( \frac{\eta_i}{k} \cdot (R_0 + V_{r_0} \cdot t_{i,0})^2 \right)} \quad \text{while} \quad R_0 = \sqrt{\frac{k \cdot e^{-\sigma \cdot \eta_0}}{\eta_0}} \quad \text{and} \quad R_i = \sqrt{\frac{k \cdot e^{-\sigma \cdot \eta_i}}{\eta_i}}
\]

Where $R_0, \eta_0, V_{r_0}$ represent values of the target at time $t = 0$ and $\theta_i, \eta_i$ represent values of the target at time $t = 1$. The same holds for the next angle, $\theta_2$.

We use the relationship, (7), between $\theta_0, \theta_1$ and $\theta_2$ and equation (5) to find $\theta_1$ and $\theta_2$ from $\theta_0$. We cannot know the value of $\theta_0$ used in equations (5) and (7) and therefore we will scan the range of $0..2*\pi$ to find which value of $\theta_0$ solves these two equations.
Legend

- **Agent and sensing zone**: Three blue circles. The agent is in the middle of them.
- **Target**: A red circle with a centered red dot.
- **Sensed target state**: An orange circle.
- **Tracked target path**: A blue straight line. Light blue is a predicted path. Dark blue is a sensed path.
- **Semi tracked target path**: A yellow line.

Note that given a $\theta_s$, the result of the equation (7) may lead to two valid solutions. This is the reason for the use of capsules. The sampling agent will leave the decision of which of the two target states is the right one to the higher levels.

Equation (5) cannot be solved symbolically therefore the sampler agent uses computational methods. The sampler agent scans the range of $\theta_s$ and looks for suitable
locations corresponding to $\theta$. Only certain angles will fit the above equation. To be more accurate, the sampler agent uses one more sample and applies the same mechanism to $\theta, \theta_2$ and $\theta_3$. Comparing the results of both cases assures accurate results. The calculated angles will be used to form a set of possible pairs of locations and a velocity of a target (i.e., a capsule). In the following algorithm we will use the notation $\text{sample}_i$ to represent a measurement $<\eta, V>$ at $t = i$.

Find $\theta_0$ function

**Input:** $sa, \text{sample}_0, \text{sample}_1, \text{sample}_2$

**Output:** $\theta_0$

minimum_diff = epsilon

min_0 = -1

For $\theta_0 = 0$ to $2\pi$ in delta steps

- calculate $\theta_1$ using $\theta_0$ by equation (7)
- calculate $\theta_2$ using $\theta_0$ by equation (7)
- diff = the difference between the left side the right side of equation (5)
  - using $\theta_1$ and $\theta_2$
- if (diff < minimum_diff)
  - minimum_diff = diff
  - min_0 = $\theta_0$

Return min_0

rawDataTransformation function

**Input:** $sa, \text{sample}_0, \text{sample}_1, \text{sample}_2, \text{sample}_3$

**Output:** target states.

$\theta_0 = \text{Find } \theta_0( sa, \text{sample}_0, \text{sample}_1, \text{sample}_2 )$

$\theta_1 = \text{Find } \theta_1( sa, \text{sample}_1, \text{sample}_2, \text{sample}_3 )$

if ($\theta_0 != -1$ & $\theta_1 != -1$)

- $\theta_3 = \text{calculate } \theta_3$ using $\theta_0$ by equation (7)
- $\theta_3^* = \text{calculate } \theta_3^*$ using $\theta_1$ by equation (7)

if (difference between $\theta_3$ and $\theta_3^*$ < epsilon$\theta$)

Return $<\overrightarrow{D}(\theta_3), \overrightarrow{V}(\theta_3)>$, $\overrightarrow{D}(-\theta_3), \overrightarrow{V}(-\theta_3)>$

else

Return null
**Capsule generation algorithm**

**Input:** $sa, sample_0, sample_1, sample_2, sample_3$

**Output:** capsule

```plaintext
targetStateSet = rawDataTransformation ($sa, sample_0, sample_1, sample_2, sample_3$)
if (targetStateSet != null)
  capsule = new Capsule()
  capsule.sa = sa
  capsule.states = targetStateSet
else
  capsule = null
Return capsule
```

**Theorem 6:**

The ANTs capsule generation algorithm time complexity is $O(1)$.

**Proof:**

While generating a capsule, the `rawDataTransformation` function uses the $Find_{\theta_0}$ function twice. The time complexity of scanning the range of all the angles from 0 to $2\pi$ in $Find_{\theta_0}$ 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 is CPU intensive. Sampler agents should apply this algorithm for every four consecutive measurements. Thus, they may have to apply it many times if they acquire many measurements or if many targets pass through its sector. Sampler agents may not have enough resources to accomplish this algorithm many times in real time. Therefore, we may consider using simpler sampling agents, i.e., with smaller detection sectors, which will reduce the computation load on a single agent. We 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 $\bar{D}_a =<200,200>$ with an orientation of 0 degrees. The sampler uses a Doppler with the characteristic $k = 1$ and $\sigma = 1$ and a maximum detection range of 100 meters (see (1)). The Doppler measures the following measurements:
<table>
<thead>
<tr>
<th>Time</th>
<th>η</th>
<th>$V_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.08E-04</td>
<td>0.014141</td>
</tr>
<tr>
<td>1</td>
<td>1.11E-04</td>
<td>0.042405</td>
</tr>
<tr>
<td>2</td>
<td>1.15E-04</td>
<td>0.070619</td>
</tr>
<tr>
<td>3</td>
<td>1.18E-04</td>
<td>0.098748</td>
</tr>
</tbody>
</table>

Scanning the range of 0..2*pi for the value of $\theta_0$ in Find$\theta_0$ function, the algorithm evaluates $\theta_1(\theta_0)$ and $\theta_2(\theta_0)$ using equation (7). At the end of the scanning loop the algorithm finds that while $\theta_0$ has the value of –0.7854 the difference between the left side and the right side of equation (5) is minimal. Doing the same to find $\theta_1$, the algorithm finds that when $\theta_1$ is –0.7654 the difference between the left side and the right side of equation (5) is minimal. The algorithm then uses equation (7) to find that $\theta_3(\theta_0)$ is –0.72547 and that $\theta_3^*(\theta_1)$ is also –0.72547. Understanding that the values of the calculated $\theta_3(\theta_0)$ and $\theta_3^*(\theta_1)$ are equal, the algorithm constructs two target states. The first is $<D(\theta_3), \bar{V}(\theta_3)>$ and the second is $<D(-\theta_3), \bar{V}(-\theta_3)>$. The value of $D(-\theta_3)$ and $\bar{V}(-\theta_3)$ is obtained by the trigonometric equations:

$D(\theta) = <R(\theta) \cdot \sin(\theta) + D_{a}.x, R(\theta) \cdot \cos(\theta) + D_{a}.y>$

$\bar{V}(\theta) = <\bar{V}_r \cdot \sin(\theta), \bar{V}_r \cdot \cos(\theta)>$

when $R(\theta)$ is attained by equation (8) and in our case $R(\theta_3) = 70.8378$.

According to our example the two target states will be $<<153,253><1,1>>$ and $<<247,253><-1,1>>$.

### 7.2. Objects and ResBy relation

DARPA’s ANTs program considers a large-scale environment where there are many mobile targets and many mobile Doppler sensors. According to this domain the goal of the DDM system is to track the targets. Each target moves in a steady velocity along a straight
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, known in physics literature as the radius vector, is the vector from the axis origin \((0,0)\) to the target coordination. The velocity vector describes how the target changes location in one second. A steady motion equation may look like this:

\[
f(t) = \langle \vec{r}_0 + \vec{v} \cdot t, \vec{v} \rangle
\]

where \(\vec{r}_0\) is the location of the target at time \(t = 0\) and \(\vec{v}\) is the velocity vector. Thus, the goal of the DDM system is to identify the motion equation of each target in the area.

Following the ANTs domain, objects correspond to targets. The target state structure is \(s = \langle \vec{r}, \vec{v} \rangle\). \(\vec{r}\) is the location vector of the target and \(\vec{v}\) is the velocity vector. If a target state \(s_2\) at \(t_2\) resulted from target state \(s_1\) at \(t_1\) and the velocity of the target remained constant during the period \(t_1, t_2\), then \(\vec{r}_2 = \vec{r}_1 + \vec{v} \cdot (t_2 - t_1)\). We assume that no target is likely to appear with the same exact properties as another target. That is, there cannot be two targets at the exact same location moving in the same velocity and direction. Thus, in ANTs where \(s_1 = \langle \vec{r}_1, \vec{v}_1 \rangle\) \(\text{ResBy}(\langle t_1, s_1 \rangle, \langle t_2, s_2 \rangle)\) is true iff: (i) \(r_2\) may be derived from \(r_1\) using the motion equation of a target and given \(\vec{v}_1\) during the period \(t_2 - t_1\) and (ii) \(\vec{v}_1 = \vec{v}_2\).

The physical motion of a moving body in a steady velocity follows the four constraints of the \(\text{ResBy}\) relation. In general, in any domain every object state that combines a singular state with the first derivative of this state by time where this derivative is not dependent on time satisfies the four constraints.

### 7.3. PosS implementation

A measurement in the ANTs domain is a pair of amplitude and radial velocity values for each sensed target. Given a measurement of a Doppler radar the target is located by the Doppler equation:

\[
r_t^2 = \frac{-\left(\beta - \chi\right)^2}{\kappa \cdot e^{\frac{\sigma}{\eta_i}}}
\]
where, for each sensed target, \( i \), \( r_i \) is the distance between the sensor and \( i \); \( \theta_i \) is the angle between the sensor and \( i \); \( \eta_i \) is the measured amplitude of \( i \); \( \beta \) is the sensor beam angle; and \( k \) and \( \sigma \) are characteristics of the sensors and influence the shape of the sensor detecting area (1). Given \( k \) consecutive measurements one can use the Doppler equation to find distance \( r_i \).

However, there are two possible \( \theta_i \) angles for each of these distances. Therefore, the PosS function in the ANTs domain returns two possible object states, i.e. \( m=2 \).

**Theorem 7: (PosS in ANTs)** Assume that the acceleration of a target in a short time period is zero. The next target location after a very short time is then derived from

\[
\theta_i = \alpha_i + \sqrt{-\sigma \cdot \ln \left( \frac{\eta_i}{k} \left( v_0 + v_{i0} \cdot t_{i0} \right)^2 \right)}
\]

\[
\frac{\bar{r}_2(\theta_i) - \bar{r}_1(\theta_i)}{t_{2,1}} = \frac{\bar{r}_1(\theta_i) - \bar{r}_0(\theta_i)}{t_{1,0}}
\]

where \( r_0, \theta_0, \eta_0, v_{i0} \) and \( \alpha_0 \) are values of the target at time \( t=0 \) and \( \theta_i, \eta_i \) and \( \alpha_i \) represent values of the target at time \( t=1 \). \( t_{i,j} \) is the time between \( t=i \) and \( t=j \).

Only certain angles will solve the equations. To be more accurate, the sampling agent uses one more sample and applies the same mechanism to \( \theta_1, \theta_2 \) and \( \theta_3 \). The angles are used to form a set of possible pairs of location and the velocity of a target (i.e., the PosS function values). Only one of these target states is the correct one.

**Definition 18:** In addition, \( pathToFunc(p) \) is calculated using the following:

\[
f_{x_{t+1},x_t}(t) = \langle \bar{r}_s - \bar{v}_e \cdot (t - t_e), \bar{v}_s \rangle \text{ which is equivalent to } \langle \bar{r}_e - \bar{v}_e \cdot (t - t_e), \bar{v}_e \rangle.
\]

### 7.4. 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 \( \tau \).
seconds before processing new data to avoid data overloading. Updating the localInfo involves three steps:

i. Obtaining new information from the leader’s subordinates;

ii. Finding new paths; and

iii. Merging the new paths into the localInfo.

Below we present 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 from their unusedCapsules and adds them to its unusedCapsules set. The zone group leader obtains localInfo from its subordinates. It adds the unusedCapsules to its unusedCapsules and merges the infoMap of that localInfo to its own localInfo.

The merging of functions is performed both in steps (i) and (iii). As I noted earlier the merging of functions is needed since target state functions which 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 different functions coming from different agents may 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.

### Step 1 - Obtaining new information algorithm

**Input:** localInfo  
**Output:** updated localInfo  

```
if activated as Sampler group leader
    for each subjugated sampler, sampler
        additionalCapsules = obtain set of capsules from each sampler
        localInfo.unusedCapsules = localInfo.unusedCapsules U additionalCapsules
else // activated in Zone group leader
    for each subjugated 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 U additionalCapsules
        mergeFunctions (localInfo.infoMap, additionalInfoMap);
return infoMap, unusedCapsules
```
**mergeFunctions algorithm**

**Input:** target function sets: \( F, F' \)

**Output:** 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

\[
\text{if } (f^j, D(0) - f^j, D(0) < \epsilon D \land f^i, V - f^i, V < \epsilon V)
\]

\[
f^j, t_s = \min(f^j, t_s, f^j, t_s)
\]

\[
f^j, t_e = \max(f^j, t_e, f^j, t_e)
\]

merged = true

if (not merged)

\[
\text{Return } F = F \cup \{f^j\}
\]

**Lemma 17:**

Let \( p^1 = <\pi^1_1, ..., \pi^1_s> \), \( p^2 = <\pi^2_1, ..., \pi^2_s> \) be two paths, where \( \pi^j = <t^j_s, s_a^j, s_j> \) and

\[
f^1_{\pi^1_1, \pi^1_s}(t) = <\pi^1_s, s_a^1, \overline{D} - \pi^1_s, \overline{V} \cdot (t - \pi^1_s, t), \pi^1_s, \overline{V} >,
\]

\[
f^2_{\pi^2_1, \pi^2_s}(t) = <\pi^2_s, s_a^2, \overline{D} - \pi^2_s, \overline{V} \cdot (t - \pi^2_s, t), \pi^2_s, \overline{V} >.
\]

if \( \text{ResBy}(<t^1_s, s^1_a>, <t^1_s, s^1_j>) \) then for any \( f^1_{\pi^1_1, \pi^1_s}(t), f^2_{\pi^2_1, \pi^2_s}(t) \)

\[
f^1_{\pi^1_1, \pi^1_s}(t) = f^2_{\pi^2_1, \pi^2_s}(t)
\]

Using lemma (17) we developed the *mergeFunctions* algorithm. In this algorithm, the leader uses the *ResBy* relation to check whether the first state of the target state function resulted from the first state of a different target state function. If one of the states is affected by 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 range of the new function is the largest possible range given the points found. In case 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.
Lemma 18:
The time complexity of the obtaining new information algorithm is $O(T^2)$ where $T$ is the number of targets in the $\tau$ seconds window of time in which target information is kept by an agent.

Proof:
While obtaining new information the algorithm asks every subjugated agent for its information. The number of subjugated agents is predefined and therefore is constant. In the case of sampler leaders the algorithm combines all capsules. Such combinations will be dependent on the number of capsules, which is correlated to the number of targets with a constant factor. The constant factor depends on predefined constant values, such as, the number of agents and the time period for sampling. Therefore the time complexity for the sampler leader part is $O(T)$. However, per each subjugated 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 per every other task state function in another set.

The second step 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 have a $\text{ResBy}$ relation with two different states of another capsule with respect to the capsule’s time.

Lemma 19:
Let $C_1 = <t^1, sa^1, <s^1, s^2_1>>$, $C_2 = <t^2, sa^2, <s^1_2, s^2_2>>$ then if $\text{ResBy} ( <t^1, s^1_1, <t^1, s^2_1>)$ then

(i) $\text{ResBy} ( <t^1, s^1_1, <t^1, s^2_1>)$ is false and

(ii) $\text{ResBy} ( <t^1, s^1_1, <t^1, s^2_1>)$ is false
Proof:

If it could have such a relationship with both target states, the two targets states should have a ResBy relation between themselves. Given that the two target states have the same creation time and that a target cannot be in two places at the same time this possibility is contradicted.

**Step 2 - Finding new paths algorithm**

**Input:** unusedCapsules  
**Output:** updated unusedCapsules, accurateFunctions, mediocrePaths

// phase 1: make links
sort(unusedCapsules) // by time stamp
allPaths = {}  
for each capsule, c=<t, sa, {s₁, s₂}>, in unusedCapsules  
  cap.mark = false  // marking for phase 2  
  for each target state, si, in cap states  
    linked = false  // because of the above assumption and given that the path  
    // elements came from capsules there will be only one suitable  
    // path. Therefore, we exit the loop after finding such a path  
    for every last triplet, <t-last, sa-last, s-last>, in each path, p,  
      in allPaths && not linked  
        if (ResBy(<t-last, sa-last>, <t, si>) or (t-last=t && sa-last ≠ sa))  
          p = p* <t, sa, si>  
          linked = true  
    if (not linked)  
      p = <t, sa, si>  
    allPaths = allPaths ∪ {p}

// phase 2: collect target representing paths that have no common capsules  
// when giving a greater priority to paths with more viewpoints.
sort(allPaths) // by number of viewpoints
paths = {}  
for each path, p, in allPaths  
  if (not isAnyCapsuleMarked(p) && numberOfViewpoint(p) > 1)  
    markAllCapsules(p)  
    unusedCapsules = unusedCapsules - allCapsules(p)  
    accurateFunction = accurateFunction ∪ {pathToFunc(p)}  
if activated as top-level leader  
  mediocrePaths = collectMediocrePaths( allPaths )  
else  
  mediocrePaths = {}  
Return unusedCapsules, accurateFunctions, mediocrePaths
In the above algorithm we create the new paths based on the capsule's information. We add two temporary fields to two of the structures only for the purpose of the algorithm. The first is a boolean flag named mark that is added to the capsule structure. The second is a pointer to the originating capsule that is added to every triplet 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 is created and the state is the first in it. In the second phase the agent separates the paths into accurate and semi accurate paths according to the number of sampling agents originating in them.

**Example:**

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

<table>
<thead>
<tr>
<th>Capsule</th>
<th>Sensing Agent State</th>
<th>Target State A</th>
<th>Target State A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Location</td>
<td>Orientation</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0,0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0,0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0,0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0,0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0,0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>10,10</td>
<td>0</td>
</tr>
</tbody>
</table>

Let us assume that at the beginning allPaths does not contain any path. Handling each target state in each capsule, we will start with TargetsStateA of capsule 0. Because allPaths does not contain any path, a new path will be created with TargetsStateA of capsule 0 at its head. The next state, TargetsStateB of capsule 0, is tested to see if it follows the ResBy with any of the paths, stored in allPaths, tails. It does not follow the ResBy when the only tail that exists is: TargetsStateA of capsule 0. Therefore, a new path will be created with TargetsStateB of capsule 0 at its head. TargetsStateA and TargetsStateB of capsule 1 will each also result in a new path. However, TargetsStateA of capsule 2 follows the ResBy with
TargetsStateA of capsule 0 and will join its path as a new tail. TargetsStateB of capsule 2 will do the same with TargetsStateB of capsule 0. At the end of the first phase 6 paths will be formed, 3 of them made from more than one capsule.

Figure 6: The ResBy relation

In the above figure we present the outcome of phase 1. Each arrow represents a ResBy relation between two path points. Every capsule contains the originating sampler state. 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 joined 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 finds semi-accurate paths. Semi-accurate paths are paths with the same point of view that have more than one target state.

The function pathToFunc receives a path and returns a function based on it. In our case it will receive the paths presented in the left side of the figure and will return:
\[ f_{\pi_1, \pi_e}(t) = << 50, 50 > + 2, 3 > t, < 2, 3 >> \] with respect to the first and the last path points:

\[ \pi_s = <0, <<0, 0>, 0>, <<50, 50>, <2, 3>> > \] and

\[ \pi_e = <3, <<10, 10>, 0>, <<56, 59>, <2, 3>> > \]

**Lemma 20:**

The time complexity of the finding new paths algorithm is \( O(T^2) \) where \( T \) is the number of targets in the \( \tau \) seconds window of time.

**Proof:**

In this algorithm, every leader runs over a set of paths per every capsule in a set of capsules. The paths and the capsule sets are correlated to the number of targets in the time period of \( \mathcal{T} \) and therefore result in a time complexity of \( O(T^2) \).
Chapter 8

Simulation, experiments and results

8.1. DDM Simulator

A simulator was developed to study large-scale problems associated with the application of DDM. The ANTs challenge was chosen as the test case of DDM and simulated Doppler radars were selected as sensors. The simulation consists of an area of a fixed size in which Doppler sensors attempt to extract the object state functions of moving targets. Windows 2000 operating system on 150 Pentium 4 based computers with 1GB Ram was used to run the simulations. A description of the DDM simulation and the behavior of its components are elaborated upon in the following paragraphs.

8.1.1. Target movement

Each target has an initial random location along the border of an area and an initial random velocity of up to 50 kilometers per hour in a direction that leads inwards. Targets leave the area when reaching the boundaries of the zone. Each target that leaves the area causes a new target to appear at a random location along the border and with a random velocity in a direction that leads inwards. Therefore, each target may remain in the area for a random time period.

8.1.2. Sensor movement

While the integration algorithms play an important role in producing an accurate solution, ultimately, the accuracy of the solution fundamentally depends on the accuracy of the observations made by the sampling sensor agents. There are several degrees of freedom...
associated with the movements of sampling sensor agents. It is, therefore, important to
determine the best way to locate them. Each sensor can be in either one of two states: (i)
sensing state and (ii) movement state. A sensor that collects information must be stationary
and a moving sensor cannot sense any target at that time. Each sensor remains within each
state for a predefined period of time. Each sensor alternates between sensing and moving. An
exception to this behavior holds when the load balancing algorithm is applied. In that case the
sampling sensor agent may also move according to the load balance algorithm. As in the
movement state, while moving according to the load balance algorithm the sensor cannot
sense targets. In a movement state or while following the load balance algorithm each
sampling sensor agent's speed limit is 50 kilometers per hour. A sampling sensor agent may
move in several ways while in the movement state. The differences between the following
types of movement are presented later in this chapter:

**Steady random movement**

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 directed into the zone.

**Rectangle and triangle patrol movement pattern**

After analyzing the problem 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 time \( t+1 \) should be minimized. One of the ways to achieve this is to move in
the pattern demonstrated in figure 7. We refer to this pattern as the *rectangle patrol
movement pattern*. Another way for a sampler agent to move is in a different pattern as
shown in figure 8. We refer to this movement pattern as the *triangle patrol movement
pattern*. 
8.1.3. The coverage density

Three sets of experiments were conducted to examine DDM. The experiment sets vary in the scale of the problem. The first set examines DDM in medium-scale environments; the second in large-scale environments and the third in very large-scale environments.

The coverage density figure, introduced in chapter 2, is used to distinguish between the three sets of experiments. In the first set the coverage density is high, in the second it is medium and in the third it is low. As the coverage density decreases the DDM solution takes on a more important role since agents should be used wisely to track objects.
<table>
<thead>
<tr>
<th>Experiment Set 1</th>
<th>Experiment Set 2</th>
<th>Experiment Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDM in a medium scale environment</td>
<td>DDM in a large-scale environment</td>
<td>DDM in a very large-scale environment</td>
</tr>
<tr>
<td>$Z$ (square meters)</td>
<td>1,080,000</td>
<td>100,000,000</td>
</tr>
</tbody>
</table>

Agent properties

| Range of interaction (meters) | 200 | 100 | 50 |
| Sensing time (seconds) | 10 | 5 | 10 |
| Moving Time (seconds) | 10 | 1 | 5 |

\[
\text{Area when sensing (square meters)} = 31,416 \\
\text{Area when moving (square meters)} = 0 \\
\text{Avg Area} \left( \text{square meters per second} \right) = \begin{cases} 15,708 & \text{set 1} \\ 6,545 & \text{set 2} \\ 1,309 & \text{set 3} \end{cases}
\]

Number of agents

| 20 | 1,000 | 5,000 |

\[
\text{Avg Area} \left( \text{square meters per second} \right) = \begin{cases} 314,159 & \text{set 1} \\ 6,544,985 & \text{set 2} \\ 6,544,985 & \text{set 3} \end{cases}
\]

\[
\rho \left( \frac{1}{\text{second}} \right) = \begin{cases} 29.09\% & \text{set 1} \\ 6.54\% & \text{set 2} \\ 1.64\% & \text{set 3} \end{cases}
\]

**Figure 9: Coverage density of the three experiment sets**

A comparison between the basic settings of the three experiment sets is presented in figure 9. In the first experiment set the basic settings simulates a 1,080,000 square meter area. There are 20 Doppler agents and each of them contains a Doppler with a beam range of 200 meters. Since the Doppler beam is essentially a circle while the range of interaction is its
diameter, the area covered by the beam is 31,416 square meters. Every Doppler senses targets for 10 seconds and then moves for 10 seconds. That is, on the average the agent coverage, \( a_i \cdot \omega \), is 15,708 square meters per second. Using 20 similar Dopplers leads to a total coverage, \( A \omega \), of 314,159 square meters per second. Dividing the total coverage by the size of the controlled zone results in a coverage density of 29.09%.

The next experiment set is much larger. In the basic settings of this experiment 1,000 agents track objects in a 100,000,000 square meter zone. Namely, the controlled area is almost 100 times larger and 500 times more agents are active. Moreover, each agent is less capable of tracking objects since its range of interaction is half that of a similar agent in experiment set 1. Decreasing the range of interaction by half means that the coverage of each agent is reduced to a quarter of the covered area. Since each Doppler senses objects for 5 seconds and moves for 1, the agent coverage is only 6,545 square meters per second. This experiment set uses agent coverage that is much smaller than the previous experiment set. Thus each agent may be less expensive since its capabilities are decreased dramatically. By using 500 times more agents we reach a total coverage of 6,544,985 square meters per second. Dividing this by the size of the controlled area shows that the coverage density of this experiment set is only 6.54% in comparison to 29.09% in the first experiment set. This coverage density reduction of 77% suggests that the system might suffer from a dramatic reduction in its performance [45]. We will show that this is not the case when using the DDM architecture.

In the last experiment set, 5,000 Doppler agents track objects across 400,000,000 square meters. This is 5 times the agents’ number and 4 times the controlled zone size than the previous set of experiments. A further reduction in the range of interaction and sensing time of 10 seconds with a moving time of only 5 leads to an agent coverage of 1,309 square meters per second. Using 5,000 Dopplers to track objects results in the same total coverage as in the previous experiment set, i.e. 6,544,985 square meters per second. Dividing the total coverage in a 4 times larger area size reflects a 1.64% coverage density. Note that the decrement of the
coverage density from 6.54% to 1.64% is 75%, which is equivalent to the decrement from a coverage density of 29.09% to 6.54%. Later in this chapter we will show how these decrements influence the performance of the DDM solution.

8.1.4. Evaluation methods

We collected the state functions produced by agents during a simulation. We used two evaluation criteria in our simulations: (i) target tracking percentage and (ii) average tracking time. We counted a target as tracked if the path identified by the agent satisfied the following: (a) the maximum distance between the calculated location and the real location of the target did not exceed 1 meter, and (b) the maximum difference between the calculated $v(t)$ vector and the real $v(t)$ vector was less than 0.1 meters per second and an angle of 0.1 radians.

A sensed target may result either in (i) an object state function that represents its behavior as a function of time or (ii) two object state functions with an equal probability to represent its behavior as a function of time.

The first type of object state functions will be considered 100% object state functions, while the second type of object state functions will be referred to as 50% object state functions. 100% object state functions will be integrated into the infoMap, while each 50% object state function pairs will be kept till one function in a pair is confirmed by supporting measurements. One set of agents will be considered to have performed better than another if it reached a higher tracking percentage with a lower tracking time with respect to the 100% functions and the total tracking percentage was at least the same.

In the first and the second experiment sets, the averages reported in the graphs below were computed for one hour of simulated time. The target tracking percentage 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. The tracking time was defined as the time period that the agents needed to find the object state function of the target from the time the target entered the simulation. The tracking average time was calculated by dividing the sum of tracking time of the tracked targets by the number of tracked targets.
In the last experiment set averages were computed for seven days of simulation time. Simulating seven days, opposed to one hour in previous experiment sets, required an adjustment of the evaluating methods. The average target tracking 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 over periods of time. The average target tracking was calculated for 10 seconds, 1 hour and 3 hour periods of time. We will show that varying the time period of the average calculation provides an indication about the performances of DDM.

In this last experiment set, only targets for which a single function was associated are reported. In a time period $t_1..t_2$, a target was reported as current tracked if the target stayed in the controlled zone during that period of time and a state's change function was associated with it at $t'$ while $t' \leq t_2$. Intuitively, usually an image will be presented in a command and control center describing the tracked targets. Current tracked targets represent the tracked targets that are being presented right now without considering the targets that already left the controlled zone. This information is the relevant information for tracking and surveillance purposes and is the main concern of the users of such systems. Since the last experiment set was simulated for 7 days in comparison to only one hour in the first two sets, this distinction was necessary for only the last experiment set. As in previous experiment sets, we will also refer to the total number of tracked targets.

8.2. Set 1: DDM in a medium scale environment

Several basic issues needed to be explored in a medium scale environment via simulations. First the ability of the DDM model to identify state functions had to be ascertained. Second, we had to prove that a simple hierarchy model improved the performance of the system. Third, the model's sensitivity to noise had to be established. Finally, we wanted to examine whether increasing the number of agents and using better equipped sampling agents would improve performance. In this experiment set, there is only one hierarchy level and therefore only one leading agent. The samplers and the leader in this
experiment set do not react to the changes in the functioning agent community and do not follow any load balancing mechanism.

8.2.1. Basic settings

The basic settings for the environment corresponded to an area of 1200 by 900 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.

Dopplers:

The Dopplers were mobile and moved randomly as described above. 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 and the number of Doppler sensors was 20.

Targets:

The number of targets at a given time point was 30. In total, 670 targets passed through the controlled area within an hour in the basic settings experiments. Note that preliminary experiments showed that 29% of the targets in this experiment set remained in the area less than 60 seconds with our basic settings.

Hierarchy:

The DDM hierarchy consisted of only one level. That is, there was one sampler-leader that was responsible for the entire area. First we compared several settings to test the hierarchy model and the sampling agents characterizations. Each setting was characterized by (i) whether a hierarchy model (H) or a flat model (F) was used; (ii) whether the sampler-agents were mobile (M) or static (S); and (iii) whether Dopplers varied their active sectors from time to time (V) or used a constant one the entire time (C). In the flat model the sampler agents used only their own capsules to produce object state functions locally. In the following paragraphs we denote a combination of the described characteristics using their code names,
i.e. FSC would be a (F) flat model, that includes only (S) static agents each of which (C) do not vary their active sectors.

The tolerance capabilities of the DDM architecture towards noisy data are discussed next. Subsequently, the load of the data transmission used to communicate between agents is surveyed. This includes the communication load constraints and how the transmission is influenced.

Later, we elaborate on the influences of reducing and raising the coverage density. We will demonstrate two different properties: (i) the number of sensors and (ii) the detection range of a sensor. Note that while the first parameter is a property of the system, the second is a property of the individual sensor. Nonetheless, the system designer may compensate for weaknesses of one property with the other, i.e. adding more sensors.

### 8.2.2. Mobile and dynamic vs. static sensors

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

### 8.2.3. Hierarchy vs. flat models

Since the next four settings represent important combinations, we focus on the following characteristics:

A. FSC that involves static Doppler sensors with a constant active sector using a nonhierarchical model;
B. HSC as in (A) but using the hierarchical model;
C. FMV with mobile Dopplers that vary their active sectors from time to time, but with no hierarchy;
D. HMV as in (C) but using the hierarchical model.

<table>
<thead>
<tr>
<th>Method</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile Doppler</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Varying sector</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Figure 10: Medium scale - Summary of methods

Method D is the basic setting, while A, B and C are the methods used to compare with D.

We tested FSC on two experimental arrangements: randomly located Dopplers and Dopplers 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 HMV would do better than the agents in all of the other settings.

The first finding is presented in Figure 11. This figure indicates that the setting does not affect the overall tracking percentage. The difference between the results of 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 faster than the others. This supports the hypothesis that a hierarchical organization leads to better performance. Further support can be in the findings showing HSC performed significantly better than FMV even though, according to our preliminary results, HSC uses Dopplers which are more primitive than the FMV Dopplers.
Another measure for evaluating the performance of these models is the average tracking time of the Doppler sensors. Figure 12 shows the differences in this amount between the approaches. Once again, one can see that hierarchically based organizations lead to better
results. We found that by considering only targets that stayed in the controlled zone at least 60 seconds, HMV reached an 87% tracking percentage where 83% were accurately detected. We also considered a hierarchy with two levels: one zone leader leading four sampling leaders. The area was divided equally between the four sampling leaders, and each obtained information from the many mobile sampling agents located in its area. In this 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 shows that there was no significant difference in the performance (with respect to the evaluation definition) of the system when there were two levels in the hierarchy as opposed to only one level in the hierarchy. However, consistent with theorem 1 (see chapter 4), the computation time of the system was much less in the case of two-level hierarchy.

8.2.4. Communication and noise

While the performance of the hierarchy-based models are significantly better than the non-hierarchical 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, then FMV should be used while considering the cost of communication. Even in situations where communication is possible, one must study the possibility of messages being lost or corrupted. In our experiments, communication focuses on the transmission of capsule data structures for possible targets. The sampling agents in our experiments transmit 168 bytes per minute.

The influence of randomly corrupted capsules on HMV’s behavior was examined. Our results indicate that up to a level of 10% noise, the performance decrease is moderate. Figure 13 shows that as the percentage of the lost capsules increased the number of tracked targets decreased; 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% resulted in a smaller decrease to a tracking accuracy of 70% while the tracking time increased slightly to 71. DDM could even manage with noise of 30% and track
39% of targets with an average tracking time of 115 seconds. This system behavior indicates that DDM is tolerant to noisy communication. In the rest of the experiments we used the HMV settings without noise.

Figure 13: Medium scale - Target detection percentage as a function of the communication noise

Figure 14: Medium scale - Target detection average time as a function of the communication noise
8.2.5. Sensor and target population

As stated in figure 9 the coverage density of the basic settings of experiment set 1 is 29.09%. We examined the effect of the number of Doppler sensors on performance and the influences of the different coverage densities. We compared the performances of the basic settings configuration with 2, 3, 5, 10, 15 and 20 Doppler sensors. That is a coverage density of 2.91%, 4.36%, 7.27%, 14.54%, 21.82% and 29.09%, respectively.

We found that when the number of targets was fixed, the percentage of targets accurately tracked increased as the number of Doppler sensors increased. The significance of this result is that it confirms that the system can make good use of additional resources. We also found that as the number of Doppler sensors increased, the 50% object state functions decreased. This may be explained by the fact that 100% object state functions result from taking into consideration more than one sample viewpoint. We also found that increasing the number of targets, while keeping the number of Dopplers fixed, does not influence the system’s performance. This is because an active sector could distinguish more than one target in that sector.

![Target Tracking](image)

**Figure 15: Medium scale - Tracking percentage as a function of the number of sensors**
8.2.6. Sensor range of interaction

In addition we tested the influence of the detecting sector area, derived from the sensor maximum range property, on performance. The basic setting uses Doppler sensors with a detection range of 200 meters. We compared the basic setting to similar ones with detection ranges of 50, 100 and 150 meters. This translates to coverage densities of 1.82%, 7.27% and 16.36%, respectively. We found that as the maximum range increased, and thus the coverage density increased, the tracking percentage increased. As the maximum radius of detection increased the tracking average time decreased. This is a beneficial property, since it indicates that better equipment will lead to better performance.

8.3. Set 2: DDM in a large scale environment

In the second set of experiments we increased the scale of the ANTS problem and the DDM solution in all dimensions. The goal of this experiment set was to investigate the
performances of the DDM solution for large scale problems, to test the influence of the number of hierarchy levels and to evaluate the fault tolerance capabilities of DDM. In this experiment set, as in the former set, samplers and leaders still did not react to the changes in the functioning agent community and did not apply any load balancing algorithm.

i. The controlled area size was increased from 1,080,000 to 100,000,000 square meters. This reflects an increase of about 100 times greater than in the first experiment set.

ii. The number of Dopplers was increased from 20 to 1,000, constituting an increase of 500 from the number used in first experiment set.

iii. The number of targets that needed to be detected was increased from 30 to 1,500, reflecting an increase of 500 times the first experiment set.

iv. The number of levels in the hierarchy was increased from one level to four levels.

Additionally, the following properties were also changed to make the problem even harder:

i. The range of interaction was decreased from 200 to 100 meters.

ii. The coverage density, $\rho$, was decreased from 29.09% to 6.54%. This reflects a decrease of 77% from the first experiment set.

8.3.1. Basic settings

The basic settings of 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.

Dopplers:

Each Doppler moved one second and stopped for five seconds to take five 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 square meters. Each Doppler sensor agent was assigned to such an area and executed the rectangle patrol movement pattern while in its movement state.

**Targets:**

The number of targets at a given time point was 1,500. In total, during one hour 4,200 targets entered and exited the controlled area at any given time. This number was counted during the simulation time since it was dependent on the size of the controlled zone, the number of targets in it at any given time and their speed limit and distribution.

**Hierarchy:**

In the basic settings 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 quarters and appointed a sub-leader to lead each one of them. Therefore 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 while each hierarchical leader communicated with the four sub-leaders directly below.

We conducted three sets of tests: (i) to evaluate the basic settings, (ii) to investigate the influences of the number of levels in the hierarchy, and (iii) to study the tolerance towards faulty sensing agents, leaders and sensing noises. In this experiment set, as mentioned earlier, samplers and leaders do not react to the changes in the functioning agent community. The following figures present a detailed study of the DDM solution for this large-scale agent system challenge. Our hypothesis was that applying the DDM hierarchy model would allow for better and faster tracking of targets. We also hypothesized that each target would be tracked for longer periods of time. We ran the simulation using the basic settings and evaluated the results.
Figure 17 depicts 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. As seen in the figure, the system accurately tracked 83% of the targets within 360 seconds. This was achieved with Doppler sensor agents covering only 8% of the area. 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 gained a fast tracking record.

Figure 18 displays the number of targets that were tracked from the time of entering a zone. Most of the tracking was achieved in less than 2 minutes from the time of the target's entrance into the zone. The system tracks 71% of its tracked targets in this period.

Figure 19 plots the tracking duration which is the time period between the first and the last time a target was detected. The figure shows that the system tracks more targets for less duration. However, it tracks most of the tracked targets for more than 6 minutes.
Figure 18: Large scale - Time to track distribution

Figure 19: Large scale - Tracking duration distribution
8.3.2. Hierarchy level

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, the more leader agents involved in generating the global solution, the less accurate the solution obtained. The hypothesis was based on the assumption that during each data fusion process a small fragment of data may get lost since the data fusion in the DDM solution follows a greedy approach. The greedy approach in DDM reflects the fact that each leader, in its turn, tries to meet its data fragments before it passes the results to its leader. In some cases a leader may not fit two data fragments right and as a result, information may get lost. As the number of data fragment is higher, so is the number of possible combinations of data fragments. This leads to a higher probability to lose information in case of a larger number of data fragments.

Figure 20: Large scale - Accurate tracked target percentage as a function of the number of levels

Figure 20 presents the tracking performances of the system as a function of the number of levels in the hierarchy. As hypothesized, the system tracked fewer 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 decrease is narrow.

Figure 21: Large scale - Accurate tracking time as a function of the number of levels

As illustrated in figure 21, the average time to track a target increases as the number of level increases. However, it increases only from 100 seconds to 106 seconds when increasing the number of levels from 1 to 4.
Figure 22: Large scale - Maximum agent process time as a function of the number of levels

Figure 22 presents the maximum time an agent needs to perform its task using the above hardware limitations. The maximum time is very close to the average time; therefore the latter is not presented here. As predicted, using one level only is insufficient to form a global solution in real time. It took the system about 4 hours to reach a solution for one simulated hour. 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 perform its task to 10 minutes.

Figure 23 displays the total amount of bytes transferred between agents during one hour when using different numbers of levels in the hierarchy. The capsules generated by samplers and sent to the sampler group leader results in a transfer rate of 4Mb. Having a massive communication load may cause a bottleneck for the 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. Figure 24
shows the significant reduction of the average communication load on the receiving agent when increasing the number of the levels.

Figure 23: Large scale - Bytes transferred as a function of number of levels

Figure 24: Large scale - Average number of bytes received by a single agent as a function of number of 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 a hierarchy of 1, 2, 3 and 4 levels. Figure 24 presents the bytes transferred divided by the number of agents.

8.3.3. Fault tolerance towards non-functioning sampling agents

To investigate the fault tolerance property of the hierarchy model in a large-scale environment we had the sampling agents stop functioning. We increased the number of damaged 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. However, the system would be able to perform well for a certain number of disabled sampling agents. The goal was to place a bound on the number of non-functioning sampling agents that the system could tolerate while still performing well.

![Figure 25: Large scale - Accurate tracked target percentage as a function of non-functioning samplers](image-url)
Figure 25 presents the accurately tracked targets percentage as a function of the number of samplers that stopped functioning. This figure shows that DDM may perform well even with 30% of non-functioning samplers. We found that increasing the number of disabled sampling agents also increases the time it takes to track targets. For example, by increasing the number of disabled sampling agents by 5% the average time it took to track a target increased by 6%.

8.3.4. Fault tolerance towards non-functioning leader agents

A second aspect of the system’s fault tolerance is the tolerance towards non-functioning leaders. In contrast to non-functioning samplers, a non-functioning 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 rectangle patrol Doppler movement pattern will result in a loss of information from half of the samplers. We hypothesized that the performance will be significantly influenced by this factor. To validate this hypothesis we conducted several simulations in which we varied the number of non-functioning sampler leaders.

![Figure 26: Large scale - Accurate tracked target percentage as a function of non-functioning first level leaders](image)
Figure 26 confirms the 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, results in a very low performance level.

**8.3.5. Fault tolerance towards noisy communication**

As 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 profound simulation testing the system while using faulty communication between samplers and leaders. We predicted that the system would be tolerant towards such noise up to a certain level.

![Figure 27: Large scale - Accurate tracked target percentage of patrol as a function of lost communication messages between samplers and leaders](image)

We found 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. The strong tolerance to failures is the direct result of the
redundancy of the number of agents and the revalidation of previous information by new data.

8.3.6. Random versus rectangle patrol movement

In this section we will show why the patrol movement pattern was chosen instead of the straightforward random movement pattern. The following figures illustrate a detailed comparison between the random and the rectangle patrol patterns. In these figures the darker bar represents the rectangle patrol pattern while the brighter bar represents the random pattern.

Our hypothesis was that by applying the rectangle patrol pattern we may achieve better results than when using the random Doppler movement. A better coverage of the controlled zone is achieved when using the patrol method and thus the system should track more targets with that method. A better coverage should also lead to a faster tracking of targets and extend the period that each target is tracked.

We will now survey a set of results illustrating the same figures as presented above with an additional comparison to the random movement pattern. We will refer to the comparing figures with the same figure identification as the original and the addition of the sign *.

Figures 17* and 18*: Large scale - Tracking percentage by time in zone and by Time to track distribution
When using the patrol pattern the system tracked more targets, as shown in figure 17*. The main difference was in tracking the short-term targets. With the patrol method the system accurately tracked 83% of the targets while it tracked only 78% with the random method. Not only did the patrol method track more targets, it also tracked them faster. More than 50% of the targets that stayed in the controlled zone less than 360 seconds were tracked when using the patrol pattern while less than 40% were tracked when using the random method. Most of the targets passing through the simulated area remained in the area less than 720 seconds. During that time the patrol method tracked more targets and therefore tracked faster.

Looking at figure 18*, both patrol and random methods achieve most of their tracking in less than 2 minutes from the time of the target's entrance into the zone: while the patrol tracks 71% of its tracked targets in this period the random tracks only 63%. Recall that the patrol method gains more tracked targets. This confirms our earlier hypothesis that using the patrol results in not only tracking more targets but in shorter racking time.

Figures 19* and 20*: Large scale - Tracking duration distribution and Accurate tracked target percentage as a function of the number of levels

Figure 19* shows that the patrol pattern also tracked targets longer than the random method. As expected, for the basic settings the patrol pattern performed better than the random method. It performed better in respect to 3 aspects: (i) it tracked more targets, (ii) it tracked them faster and (ii) it tracked them longer. Figure 20* presents the tracking performances of both methods as a function of the number of levels in the hierarchy. Comparing the accurate tracking percentage of the patrol and the random methods proves, once again, that the patrol movement tracked more targets regardless of the number of levels.
Figures 21* and 22*: Large scale - Accurate tracking time as a function of the number of levels and Maximum agent process time as a function of the number of levels

As shown in figure 21*, the patrol movement method also tracked the targets faster than the random method regardless of the number of levels. Moreover, as anticipated, as the number of levels decreases the system tracks targets faster. Looking at figure 22*, we assume that the patrol pattern needed more time than the random because it tracked more targets. When we divided the maximum agent process time presented by the number of tracked targets, the patrol and the random graphs almost overlapped.

Figures 25* and 26*: Large scale - Accurate tracked target percentage as a function of non-functioning samplers and Accurate tracked target percentage as a function of non-functioning first level leaders

In figure 25*, once again, we can see that the patrol method consistently tracks more targets regardless of the number of the non-functioning sampling sensors.
Figure 26* shows that the patrol pattern could tolerate a reduction of up to 13% in the number of functioning sampler leaders and yet maintain a high percentage of tracked objects. A reduction of 18% or more, results in a very low performance level. The random pattern, on the other hand, was much more tolerant to the same reduction. However, despite the fact that the patrol pattern resulted in a poor tracking percentage for high-rate non-functioning sampler leaders, the patrol movement pattern tracked targets faster.

To conclude the comparison section, we have shown that as long as leader agents are functioning, the patrol pattern performs better than the random method, as it is optimum in terms of overlapping coverage. However, the performances of the random pattern are very close to those of the patrol. On the other hand, the random pattern is much more tolerant towards non-functioning leader agents.

8.4. **Set 3: DDM in a very large scale environment**

In the last set of experiments we investigated the performance of the DDM model in a larger scale environment and a very low coverage density. We further expanded the parameters of the ANTS problem and the DDM solution with reference to three dimensions:

i. The controlled area size was increased from 100,000,000 to 400,000,000 square meters.

ii. The number of Dopplers was increased from 1,000 to 5,000.

iii. The number of levels in the hierarchy was increased from four to five hierarchy levels.

In addition, properties were further changed to make the problem even harder than in experiment set 2:

i. The range of interaction was decreased from 100 to 50 meters.

ii. The coverage density, $\rho$, was decreased from 6.54% to 1.64%. This reflects a decrease of 75% from the second experiment set.
In this experiment set we simulated 7 days of target tracking as opposed to 1 hour in the previous experiment sets. Simulation of longer hours improves the quality of the results since the initial positions of the targets have a lower impact on the results. We investigated the effects of the DDM load balancing mechanism. In the previous experiments we created a random, and thus an even distribution of targets. To investigate the load balancing algorithm, the targets distribution had to be changed since the load balance algorithm is aimed to gain better performances in an uneven distribution of targets.

An unbalanced load of targets may occur when many targets choose to move through an arbitrary zone. To study the load balance algorithm we chose to have all the targets move through the center of the controlled zone to reduce the effects of the boundaries.

To achieve a distribution every target should pass during its linear movement through a closed zone in the center of the controlled area. We arbitrarily chose the middle zone to constitute the size of four sampling leader’s subzones. Since every target must go through this area, at any given time the number of targets per square meter in this zone is likely to be greater than in any other place in the controlled zone. To balance the system, agents’ distribution should meet the distribution of the targets.

8.4.1. Basic settings

The basic setting for the environment corresponded to an area of 20,000 by 20,000 meters. In each experiment, we varied one of the parameters of the environment, keeping the other environment parameters the same as in the basic settings.

Dopplers:

Each Doppler moved for five seconds and stopped for ten seconds to take ten measurements. The maximum detection range of a Doppler in the basic setting was 50 meters; the number of Dopplers was 5,000. The controlled area was divided into 256 equal rectangles, each 1,250 x 1,250 square meters. Each patrolling Doppler was assigned to such an area and executed the triangle patrol movement pattern.
Targets:

The number of targets at a given time point was 1,000. In total, during 7 days 13,635 targets entered and exited the controlled area at any given time.

Hierarchy:

In the basic settings we used a hierarchy of 5 levels: four levels of zone group leaders and one of sampler group leaders. Each of the zone group leaders divided its zone into 4 quarters and appointed a sub-leader to lead each one of them. Therefore there was one leader at the top level of the hierarchy, 4 at the second level, 16 at the third, 64 at the fourth and 256 at the fifth. Each Doppler sensor communicated with one of the fifth-level leaders.

Load Balance:

The basic settings scenario includes activation of the DDM load balance mechanism. Since every sampling group leader controls an area of 1,250 by 1,250 meters, targets have to pass through the middle square zone in the size of 5,000 by 5000 meters.

An examination of different aspects of the large scale problem are presented in the following sections. We begin by investigating basic settings with and without using load balancing mechanisms. Later, we study the main properties of the basic settings. In each section we change only one property to study how it affects the performances of the DDM solution.

In the second section, we change the number of Doppler sensors to examine the influences of the number of agents. Then we compare different types of sensors. We use Doppler sensors with different ranges of detection to determine how the system performs under different hardware costs. In this section we show how to compensate for having cheaper sensing equipment by operating more sensors. The designer will have a choice of which alternative is less expensive.

In the next section we study another system property, termed: moving and sensing time ratio. We show how decreasing this ratio has an impact on system performance. We also show the alternative settings that produce the same results as the decreased ratio. In the fifth section we describe the affect of using different numbers of hierarchy levels. To conclude we
present a comparison of the two types of agent patrol movements: the rectangle and the triangle patrol movements. In the subsequent paragraphs we use the notations LB and NLB. LB refers to simulations that use the load balance algorithms while NLB refers to those that do not use the load balance algorithm.

In the LB basic settings, 5,000 Doppler sensor agents tracked targets, followed the triangle patrol movement and moved according to the load balance algorithm to balance the agent per target ratio. In this setting an average Doppler sensor agent used 65% of its time to sense targets, 32% of its time was used to patrol and 3% of the time was spent moving from one place to another according to the load balancing algorithm. Recalling figure 9, the sensing time of the basic settings is 10 while the moving time is 5. This is a ratio of 2:1. Even though the load balancing took 3% of each sensing agent's time the ratio between the sensing and the moving time period is maintained.

The NLB basic settings do not include a load balancing mechanism. In these settings 5,000 Dopplers follow the triangle patrol movement pattern and track targets. Subsequently, an average Doppler used 67% of its time to sense targets while the remaining 33% of its time was used to patrol. Once again the ratio between the sensing and the moving periods of time is measured as expected while keeping the 2:1 predefined ratio.

### 8.4.2. LB basic settings

Figures 28, 29 and 30 present the results of the load balancing algorithm. We present the average percentage of current targets tracked as a function of the simulated time. Figure 28 shows the average for 10 second periods, figure 29 for 1 hour and figure 30 for 3 hours. We can notice that the graphs converge after a starting period of stabilization. This period of time is required to recover from the starting configuration. The period of reaching stability is almost one day. From henceforth we refer to the performances of DDM after the period of stabilization. The figures present the current tracked targets.

Figure 28 shows that 85% of the targets were current tracked targets. Calculating the average current tracked targets for 1 hour periods results an increase of the current tracked targets. According to figure 29, for averages of 1 hour periods, the current tracked targets
percentage increased from 85% to 88%. Similar behavior occurs when increasing the average period of time from 1 hour to 3 hours. Figure 30 presents the average number of tracked targets during a 3 hour period. According to this figure the current tracked targets percentage increased from 88% to 91%. The increment of the current tracked targets percentage as a function of the average period of time is presented in figure 31.

This phenomenon is predictable and is a result of the differences between the periods of time targets spend in the controlled zone. Since each target has its own velocity and direction, there is a variance between the times they spend in the controlled zone; this variance leads to the reported results.

Figure 28: Very large scale - Basic settings with load balance - average target tracking - 10 sec.
Figure 29: Very large scale - Basic settings with load balance - average target tracking - 1 hour

Figure 30: Very large scale - Basic settings with load balance - average target tracking - 3 Hours
Figure 31: Very large scale - Basic settings with load balance - average target tracking comparison

Figure 32: Very large scale - Basic settings with load balance - time to track comparison

The time it took to track an average target from the time it entered the controlled zone is presented in figure 32. We can see that for an average of a 10 second time period, it took an hour to track targets that were reported as current tracked targets. For an average of 1 hour
time periods, it took 2,700 seconds to track a current tracked target. For an average of a 3 hours time period, it took 1,900 seconds to track a current tracked target. Averaging over different periods of time influences the reported results. As the average time period increases, the time to track current tracked targets decreases. We also found that the overall tracked target percentage while using the load balance mechanism was 98% and the average time to track target was 1410 seconds.

8.4.3. NLB basic settings

The following figures present the results of the DDM basic settings without activating the load balance algorithm. Figures 33, 34 and 35 show the percentage of average target tracking as a function of simulated time for a period of 10 seconds, 1 hour and 3 hours, respectively. As in the LB case, the behavior of DDM converges after a period of time of less than one day. Figure 36 presents the converged results. According to these results as the average period of time increases the current target tracking increases. When the average period was 10 seconds the average tracking time for a current tracked target was 87%. When the average period was 1 hour it decreased to 85.5% and when it was 3 hours it further decreased to 85.1%.
Figure 33: Very large scale - Basic settings without load balance – average target tracking - 10 sec.

Figure 34: Very large scale - Basic settings without load balance – average target tracking - 1 Hour
Figure 35: Very large scale - Basic settings without load balance – average target tracking - 3 Hours

Figure 36: Very large scale - Basic settings without load balance – average target tracking comparison
Following figure 37 we can see that as the average time period increased the average of the time to track current targets decreased. We also found that the overall tracked target percentage without the load balance mechanism was 85% and the average time to track a target was 981 seconds. In the next paragraph we will focus on the overall results of the tracked target percentage and the overall average time to track a target.

**8.4.4. Overall LB vs. NLB**

Figure 38 illustrates the comparison between the LB overall average target tracking percentage and the NLB results. According to this figure we can see that the load balance algorithm dramatically improves the percentage of the tracked targets.
Figure 38: Very large scale - Basic settings – overall average target detection comparison

The cost of improving the amount of the tracked targets by applying a load balancing mechanism is shown in figure 39. The time it took to track an average target while using the load balance algorithm was 40% more than without it. This cost was predicted and can be clarified by the following explanation. In the basic settings the load of a target in the center of the controlled zone is much greater than in any other place. This load leads to a great concentration of sensing agents in the middle of the controlled zone when using the load balance algorithm. Let us recall that the time to track a target is defined as the time period since the target entered the controlled zone until it is tracked. Given that, we can now claim that the probability to track targets is much greater in the center of the controlled zone. The period of time since a target enters the controlled zone until it is tracked near the center of the controlled zone is significant due to the large scale property of the problem. When not using the load balance algorithm, the sensing agents are spread uniformly around the controlled zone and have a greater chance of tracking targets as soon as they enter the controlled zone.
The distribution of the time to track an average target is presented in figures 40 and 41. According to the distributions one may see that it takes more time to track targets with the load balance algorithm.
In the following paragraphs we discuss the affects of certain DDM characteristics. These DDM characteristics represent characteristics in every similar large scale problem. We will discuss the overall tracking percentage and the overall average time to track a target. We compare the results to those of the basic settings. In the following figures the results of the basic setting are denoted in bold symbols.

As stated earlier, the coverage density $\rho$ characterizes the scale of a given large scale agent system problem (see chapter 3). The more $\rho$ decreases the greater the problem. In the following sections we present a study on the properties that affect $\rho$. By influencing the coverage density we will show the different alternatives that can achieve the same results.

We will begin by changing the number of sensing agents. As the number of sensing agents increases, the coverage density grows. The number of sensing agents not only influences the coverage density but also holds an important role in distributing the solution and reducing the computation load. We will show that up to a certain point, decreasing the coverage density by reducing the number of sensors slightly affects the amount of tracked targets. In our load balancing algorithm case, having only 3000 sensors yields almost equivalent results to situations with a much larger number of sensors. Having only 3000 sensors moderately reduces the tracked target percentage in comparison to not using the load.
balance algorithm. Using fewer sensors will reduce the tracked target percentage more dramatically.

After focusing on the impact changing the quantity of sensors has on the system, we proceeded to study the impact of changing the quality of the sensors. The range of interaction in the basic settings was 50 meters. In the following section we compare the behavior of DDM while using sensors with different ranges of interaction. This range reflects the complexity and the cost of the sensors. As the range of interaction increases, the sensor is likely to be more complex and expensive. Therefore, there may be an interest to use sensors with smaller ranges of interaction. However, increasing the range of interaction reduces the coverage density and therefore increases the scale of the problem. In this section we will show how decreasing the coverage density by reducing the range of interaction influences DDM performance with and without applying the load balance algorithm.

The next property affecting the coverage density is the ratio between the sensing time and the moving time. As in the range of interaction, the ratio affects the cost of using DDM. Activating the sensor for longer periods of time is likely to cost more if the energy of producing the electromagnetic beam is expensive. On the other hand, a more mobile sensor may cost more if the cost of the fuel needed to drive a sensor around the controlled zone is expensive. Activating the sensor for longer periods of time at the expense of movement time decreases the coverage density and therefore reduces the scale of the problem. In this section we will show that this period of time influences the performance of DDM with and without activating the load balance mechanism. We will also show how to compensate on using simple and less expensive sensors by increasing the period of the sensing time.

The next section will focus on the hierarchic structure. We will present the cost of distributing the computation by increasing the number of levels. We will show that up to a certain number of levels the performances of DDM while using the load balance algorithm remains unharmed. Increasing the number of levels up to this point supports greater distribution of the computation load. Not using the load balancing algorithm does not yield the same result. In this case any increment in the number of levels of the hierarchy leads to tracking less targets.

In the last section we will compare the two different sensor movement patterns: the rectangular and the triangular movement patterns (see chapter 8.1.2). DDM will use each
pattern with and without activating the load balance algorithm. Discussing the different aspects of the movement pattern will lead to a conclusion as to which movement pattern to use for each case.

### 8.4.5. Sensor population

After examining the basic settings, we investigated the influence of the number of sensor agents on the performance of the DDM solution. During this investigation we ran 5 different scenarios. Each scenario had the same properties with a different number of sensors and was simulated 7 days. In the first scenario there were 2,000 sensors; in the second there were 3,000 sensors; in the third there were 4,000 sensors; in the forth there were 5,000 sensors and in the fifth there were 9,000 sensors. The coverage densities of the scenarios were 0.65%, 0.98%, 1.31%, 1.64% and 2.95%, respectively, whereas the basic setting trial had a coverage density of 1.64.

This investigation supported the analysis of the previous experiment settings. In the previous experiment settings we saw an improvement of the performances as the number of sensors increased. The same influence was found in this experiment setting. The significance of this result is that it confirms that the system can efficiently utilize additional resources regardless of the degree of the problem.

As can be seen in figure 42, while applying the load balance algorithm DDM and using 3,000 sensors over an area of 400,000,000 square meters, DDM achieved 92% for overall detected targets. Using fewer sensors results in a dramatic decrease in the target detection percentage. For example, when using only 1000 sensors DDM detects 49% of the overall tracked targets.

The dramatic decrement was experienced when using 3000 sensors, in which case the system had a coverage density of 0.98%. In the next sections we will show that this decrement is a result of the coverage density and not a result of the number of sensors.
Without the load balance algorithm, DDM succeeded tracking more than 95% of the targets when it had more than 5000 sensors. Figure 43 illustrates the relationship between the number of sensors and the percentage of tracked targets.
To compare the LB and the NLB basic settings results look at figures 44 and 45. Figure 44 depicts the overall tracking percentage when using load balancing versus the overall tracking percentage without load balancing. The region between the circled and the squared graphs is the improvement gained by applying the load balance algorithm. We can see that with up to 9000 sensors the load balancing algorithm increases the number of tracked targets. When using 9000 sensors both configurations succeeded in tracking all of the targets passing through the controlled zone.

In addition to an improvement in the total target tracking percentage, there is a tradeoff when using the load balance algorithm. This tradeoff is illustrated in figure 45. As we stated (see ‘LB vs. NLB’ section) the time to track a target is significantly increased. Here, the load balance algorithm may result in an increase of 170% of the tracking time when using 9000 sensors. When applying the DDM architecture, one must take into consideration whether he wants to maximize the number of targets tracked and activate the load balancing algorithm or to minimize the time of tracking and not activate it.
8.4.6. Sensor range of interaction

Another property influencing the coverage density is the sensor maximum range of interaction. In this part of the study we varied the range of interaction and compared the performances of both with and without load balance. In different settings, the maximum detection range of each sensor was 13, 25, 50, 100 and 200 meters. This translates to coverage densities of 0.11%, 0.41%, 1.64%, 6.54% and 26.18%, respectively.

Figure 46 presents the influence of the maximum detection range of a sensor over the tracking percentage. One can see that while applying the load balance algorithm the system tracked almost all the targets for sensors with a 50 meter maximum detection range. However, without the load balance algorithm, we succeeded in attaining the same results only with sensors with a 100 meters maximum detection range. Thus to achieve the same result better sensors must to be used, which are probably much more expensive.
Figure 46: Very large scale - Sensor range of interaction: LB versus NLB average target detection

In the range of 20 to 100 meters of interaction the load balance algorithm outperforms the regular system performance. However, the cost for this better performance is shown in figure 47. The time to track a target as a function of the maximum detection range is illustrated, showing a delay in the average tracking time when using the load balance algorithm. However, below the range of about 40 meters the average tracking time without the load balance is lower. That is because for each tracked target, DDM needs two different measurements of the target state. This will mainly be a result of two sensors sensing the same target. In the case of a low detection range, there is a small probability for two sensors to sense the same target and thus tracking takes a long period of time.
8.4.7. Sensor sensing and moving time

In the basic settings each sensor repeats the following activities: (i) it senses targets for 5 seconds and then (ii) it moves for 10 seconds either in a patrol movement pattern or according to the load balance algorithm. The ratio between the moving time and the sensing time is therefore 1:2 in the basic setting case. In this case the coverage density was 1.64%. To check the impact of the coverage density on the ratio between the sensing and the moving times we compared the basic settings to settings with different sensing and moving times. We increased the sensing period to 20 seconds while the moving period remained at 10 seconds. The ratio between the sensing time and the moving time in this case was 2:1 and the coverage density was 0.82%. The fact that the coverage density in this setting was lower than for the basic settings suggests that the problem of accurately identifying the targets' trajectories is harder.

Figures 48 and 49 present a comparison between the basic settings with a moving-sensing time ratio of 1:2 and settings with a ratio of 2:1. Figure 48 describes the average target tracking of both settings with and without applying the load balance algorithm. As one can see, increasing the ratio results in a decrease in the percentage of the tracked targets both
for LB and NLB. However, the performance decrease when applying the load balance algorithm is more moderate than without the load balance algorithm. To be specific, increasing the ratio from 1:2 to 2:1 reduces the tracking percentage from 98% to 90% with the load balance algorithm and from 85% to 50% without the load balancing algorithm.

![Average Target Detection](image)

**Figure 48: Very large scale - Moving/Sensing time: LB versus NLB average target detection**

The time to track a target is presented in figure 49. This figure supports the findings of the reduction in the performances as shown in figure 48. Here one can see that the ratio change resulted in an increase in the time it took to track a target. As in the previous findings, using the load balance algorithm contributed to moderate the reduction of the performances. To be specific, the time to track a target increased from 1410 seconds to 2370 seconds when using the load balance algorithm and from 981 seconds to 2631 seconds when not using it. This reflects an increase of 170% and 270% from the basic settings results. The moderation of the increase when using the load balance algorithm has proven to be dramatic. The same phenomenon was observed when studying the influences of changing the sensor’s range of interaction. As in the former case, the reason for that is that two different measurements of the same target are needed to track the target. When the moving per sensing ratio increased...
above a certain level, the system had a very low probability of finding two different measurements describing the same target.

Figure 49: Very large scale - Moving/Sensing time: LB versus NLB time to track.

8.4.8. Hierarchy number of level

As stated in previous chapters, the large scale nature of these problems cannot be solved by a centralized approach. The DDM solution in the limited case of only one level of hierarchy is equivalent to a centralized approach. Using more levels in the hierarchy is the DDM way to allow distribution of the solution and concurrent processing by different agents. We investigated the effects of increasing the number of levels in the DDM hierarchy and the results are presented below.

Following the guidelines stated earlier each group leader leads four other groups down until the lower level that communicates with the sensing agents. The number four was chosen arbitrarily and is just an example for zone partition. The number of groups leading agents in each level is $4^{l-1}$, whereas $l$ is the level index. Consequently, the total number of leaders is the sum of all the leading agents of each level: $\sum_{l} 4^{l-1}$. For instance, a hierarchy of
3 levels involves 21 group leader agents and a hierarchy of 6 levels involves 1365 group leader agents. Such distribution of the information fusion process leads to a loss of information, as will be shown next.

Figure 50 presents the impact of the number of levels on the tracking percentage. We can see that increasing the number of levels from 3 to 5 barely affects the performance of the DDM tracking when using the load balance algorithm. However, adding one more level results in a decrease of 1% of tracked targets. Without the load balancing algorithm, the impact of distributing the solution is a little more dramatic. In this case the tracked targets percentage drops by 3% from 87% to 84%. A similar impact is presented in figure 51 where we can see that the time to track a target goes almost unchanged with the number of levels. This behavior was predicted (see *Hierarchy level* section in this chapter) since a great deal of the information fusion is accomplished in the lower levels when the load balance algorithm is applied. Fusion of basic information fragments that is done at a high level may be less successful since the data fragments need to be checked against misleading and irrelevant data fragments from a far distance. In the LB case, many sensing agents sample the same targets in a small part of the area. In our case this area is primarily the center of the controlled zone. It is likely that the immediate group leader may conclude that some of the sensed samples belong to the same target. If the immediate group leader does not do information fusion it is likely that its group leader will do so. Figure 50 and 51 also emphasize the benefits of applying the load balance algorithm. The gap between the squared and the circled graphs is the tracked target enhancement of the load balance algorithm.

Although the performances of the DDM are not improved with the number of levels, one should consider adding more levels to reduce the computation load on agents and to increase the robustness of the architecture.
Figure 50: Very large scale - Hierarchy levels: LB versus NLB average target detection

Figure 51: Very large scale - Hierarchy levels: LB versus NLB time to track
8.4.9. **Rectangular versus triangular movement pattern**

To check whether the sensor patrol movement pattern has an impact on DDM performance we compared the two patrol movements described in section 8.1.2. Comparing the results of the two movement patterns either with or without the load balance algorithm shows that there is no significant difference. Figure 52 illustrates that the target tracking percentage is the same. Figure 53 shows that the time to track a target is also the same. This can be explained by the fact that the overall coverage of the agents remained the same regardless of the movement pattern.

![Figure 52](image-url)

**Figure 52:** Very large scale - Movement pattern: LB versus NLB average target detection
8.4.10. Coverage density

To establish the importance of the coverage density definition we compared the tracking percentage of different settings. We chose the settings that produce a variety of coverage density values. Figure 54 presents the tracking percentage as a function of the coverage density for the LB and for the NLB configurations. The results are the aggregation of scenario sets of the following different properties: (i) the number of sensors; (ii) the sensor range of interaction; and (iii) the ratio between the sensor moving and sensing time.

Looking at figure 54 one cannot distinguish between the different scenario sets. The results of all the sets are situated along the curve of the LB and the NLB graphs. This proves that the important characteristic of large scale agent systems is the coverage density and not a single component of it.

To further depict the uses of the coverage density, figure 55 presents the different alternatives for the basic settings. The figure illustrates an isogram surface of a continuous variation of the three property dimensions. Recalling that the coverage density is correlated to the multiplication of these three properties (see The coverage density), each point on the
surface represents a certain number of sensors, an interaction range and a moving-sensing ratio that result in the same coverage density of 1.64%, which is the same coverage density of the basic settings. We have proven that in a large scale environment the DDM solution archives the same results for different configurations having the same coverage density.

Figure 54: Very large scale - Tracking percentage as a function of the coverage density

Figure 55: Very large scale - Alternative settings for the basic settings
8.5. **Summary of Results**

In this chapter we have presented the DDM simulator, built to simulate the DDM solution in large-scale agent systems. Using the coverage density to classify different levels of difficulty within large scale agent systems problems, we constructed three sets of experiments. The first experiment set is designated to validate the basic concepts behind the DDM solution. The second set demonstrates the DDM solution for large scale agent systems and its fault tolerance properties. The third and the main set illustrates the DDM solution for harder large scale agent systems, proving the importance of the coverage density and presents the benefits of applying the load balancing mechanism. This section summarizes the experiment findings.

**DDM in a medium scale environment:** The results reported for the first set of experiments support the following conclusions:

i. The hierarchy model outperforms the flat one.

ii. The flat mobile dynamic sector setting can be used in situations where communication is not possible.

iii. Increasing resources increases performance.

iv. Under the identified constraints, it is beneficial to add more levels to the hierarchy.

v. The DDM can handle situations of noisy communications.

**DDM in a large-scale environment:** The results reported for the second set of experiments showed that:

i. Large-scale problems involving hundreds and thousands of agents and objects cannot be solved with the traditional flat architecture while it can be solved by applying the DDM hierarchic solution.

ii. The autonomous triangle and rectangle movement patterns implemented by each sampling agent outperform the random movement method.
iii. The number of levels in the hierarchy influences the accuracy of the results. As the number of levels increases the number of tracked targets drops, even though this drop is moderate.

iv. However, as the number of level increases the time every agent needs to complete its mission drops exponentially. By combining these two results we can balance between these two properties to achieve the desired results. 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 we increase the number of levels in the hierarchy.

v. The results of the second set of experiments have established the hypothesis that, as long as leader agents continue to function, the patrol movement pattern will perform better than the random method as it is optimum. However, the performances of the random pattern are very close to optimal [57]. On the other hand, the random pattern is much more tolerant towards non-functioning leader agents.

**DDM in a very large-scale environment:** The last set of experiments yielded the following insights:

i. The environment properties such as the size of the controlled zone and the capabilities of the sensing agents are as important as the number of agents. These properties contribute to the scaling of the problem.

ii. The DDM solution is proven as an efficient solution for hard large-scale problems.

iii. The coverage density figure is proven to be a good tool for classification of hardness of large scale agent problems.

iv. Applying the load balancing algorithm improves the number of tracked targets.

v. As shown in the second set of experiments, further distribution of the solution by increasing the hierarchy height results in a small decrease in the quality of
performance. One should consider paying this cost to reduce the computation load.

vi. It takes more time to track targets when using the load balancing algorithm.

vii. The movement pattern shape is irrelevant as long as the guidelines for designing such a pattern, as described in 8.1.2, are maintained.
Chapter 9
Conclusions

This thesis significantly contributes to large scale systems with reference to the following three aspects: (i) the coverage density definition; (ii) the DDM generic architecture and (iii) the hierarchic load balancing mechanism. Each part was extensively and empirically tested through simulation of thousands of agents in order to establish reliable findings.

Coverage density, as stated in chapter 2, defines the time needed to cover an area equal to the size of the controlled zone. We have shown that there is a correlation between the coverage density of a system and its behavior. In comparing large scale systems having different coverage densities, we have proven that system properties such as the number of sensors, the range of detection of each sensor and sensor activation time have the same influence on the number of detected objects. For instance, by analyzing only the number of objects a large scale agent system successfully detects, one may not know whether there is a large number of cheap sensors or a small number of expensive ones. Given this fact, we introduced a way to achieve the same system results with different preferences. As a result, a system designer may find it easier to achieve certain system performance under given specific constraints, such as budget limits.

A hierarchical approach for combining local and partial information of large-scale objects and team environments where agents must identify the changing states of the objects, i.e. the DDM generic architecture, has been introduced. To apply this model in different environments, it is only necessary to represent three domain-specific functions: PosS, that maps measurements to possible states; ResBy, that determines whether one given object state associated with a time point can be the consequence of another given object state associated with an earlier time point; and pathToFunc, that, given a path, returns a function to represent it. Given these functions, all of the DDM algorithms implemented for the ANTS domain are applicable, as long as the complexity of these functions can be kept low. Thus, the results obtained from the ANTS simulations can be applied to any domain answering these requirements. An example of such domain can be profiling Internet users. In this case the
PosS should describe possible alternatives for an observation, ResBy should describe the connection between two observation possible descriptions and pathToFunc should bind a group of observation possible descriptions.

We have shown that such problems involving hundreds and thousands of Dopplers and targets cannot be solved with the traditional flat architecture. The novel approach presented in this thesis distributes the solution into smaller fragments of problems that can be solved partially by simple agents. 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. Ways to combine partial solutions to form a global solution have been suggested. We have established an autonomous movement algorithm to be implemented by each sampling agent. The results presented in this study show that the capabilities of the hierarchical model are greater than that of the flat one. In particular, the flat model could not solve the problem addressed in this study.

Furthermore the findings show that the number of levels in the hierarchy influences the accuracy of the results. We found that as the number of levels increases the number of tracked goals drops, albeit moderately. However, as the number of levels increase the time every agent needs to complete its mission drops exponentially. By combining these two results we are able to balance between these two properties to achieve the desired results. Choosing the right number of levels should also take into consideration the time it takes to track goals. As shown, it takes more time to track objects as the number of levels in the hierarchy increases.

In conclusion, the DDM generic architecture can perform well even if agents are very simple and inaccurate. The findings show that partial information can be combined and the existence of non-functioning participants can be overcome.

Domains may exist when agents and goals are not evenly distributed. In such situations, a load balancing mechanism is required to balance the ratio between agents and goals across the controlled zone. We have presented a novel load balance mechanism, the hierarchic load balance mechanism, which balances the sensors per goal ratio in each hierarchy level. The extensive large-scale empirical results show that the load balance algorithm increases the number of tracked goals. The cost of applying this approach was
analyzed in terms of the increase in the average time needed to track a goal. In conclusion our findings ascertain that the DDM architecture and the load balance algorithm are generic, robust and scalable.

A variety of questions emerged during the implementation of our current work which warrant further investigation and which we would like to expand upon in the future.

- We wish to demonstrate the coverage density tool on other systems and compare the results.
- We would like to affirm our hypothesis that DDM is suitable to manage systems with millions of agents and that the findings for lower scale systems also apply to much larger systems.
- We would like to investigate systems with lower coverage density. We have hypothesized that the system will perform well up to a certain coverage density value. Beyond that value the system will demonstrate very poor results. The predicated phase transition is important because of the coverage density increment cost. Finding the exact point where the system starts performing well should help achieve more efficient large scale systems.
- We would like to study how the DDM may dynamically change its architecture to achieve specified performance demands. This will allow one to commit to a quality of service.
- We would also like to study how to preserve resources by dynamically changing the system’s properties. In this case we may deactivate sensors until there is an increment in the number of objects.
- We would also like to investigate how self interest teams may take part in the DDM, how they can be motivated in large scale environments and to what degree they will contribute to common interests.
- As a large scale solution, the load balance mechanism should be further studied to learn the relation between its many properties while using the coverage density tool. This type of study could reveal how to fine tune large scale agent systems.
• To avoid computation overload in a few zones while applying the load balance algorithm, we need to study how to use zones with different sizes and how to change their size dynamically.

• Finally, another aspect that we would like to investigate is the affect of external influences on system characteristics. Systems, such as large scale supply chain management systems, involve objects and resources that may join the system at a rate and quality that is dependent on external entities. In these cases the system should adapt its other properties in order to maintain the same coverage density and thus the required performances.
Bibliography


