

**A Distributed Approach to Dynamic Autonomous Agent
Placement for Tracking Moving Targets with Application to
Monitoring Urban Environments**

A Thesis
Presented to
The Academic Faculty

by

Tamir A. Hegazy

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
November 2004

Copyright © 2004 by Tamir A. Hegazy

**A Distributed Approach to Dynamic Autonomous Agent
Placement for Tracking Moving Targets with Application to
Monitoring Urban Environments**

Approved by:

Dr. Linda Wills, Committee Chair

Dr. Aaron Lanterman

Dr. George Vachtsevanos, Advisor

Dr. Jay Lee

Dr. Bonnie Heck

Date Approved: November 15, 2004

DEDICATION

To my parents:

Abdelghany Hegazy & Houriya Gad

ACKNOWLEDGEMENTS

- Special thanks go to my family in Egypt, especially my parents. No words can describe what you have been doing for me, and no words can describe how much I love you all.
- I would like to acknowledge the support of the National Science Foundation under grant number CCR-0209179.
- I would like to sincerely thank my advisor, Dr. George Vachtsevanos, for his constant help and support throughout the doctoral program as an advisor and a father figure. Without your support and advice, I would not have gotten this work done.
- I would also like to thank all of my committee members, especially Dr. Linda Wills and Dr. Bonnie Heck for their advice and for their comments and suggestions regarding the content of this thesis.
- I would like to especially thank Dr. Jay Lee for closely following the progress of my research work, for providing valuable comments and suggestions that contributed to the quality of this thesis, and for giving me the opportunity to present my work in several occasions to the systems engineering research crowd.
- I would like to thank all current and former members of the Intelligent Control Systems Laboratory's research and administrative teams, who gave me their support and help throughout the doctoral program. With your company and support, I was able to get this done.
- Finally, I would like to thank all of my close friends. You have always been there for me to provide me with your invariable help and support, especially when times got rough. You also made the journey full of fun!

TABLE OF CONTENTS

| | |
|---|------------|
| DEDICATION | iii |
| ACKNOWLEDGEMENTS | iv |
| LIST OF FIGURES | vii |
| LIST OF SYMBOLS AND ABBREVIATIONS | ix |
| SUMMARY | xii |
| CHAPTER I INTRODUCTION | 1 |
| 1.1 Motivation | 1 |
| 1.2 Agent Placement versus Target Tracking | 2 |
| 1.3 Surveillance for Urban Warfare | 3 |
| 1.3.1 Hierarchical Decomposition of Surveillance Missions | 3 |
| 1.3.2 Surveillance in Urban Zones | 5 |
| 1.4 Thesis Organization | 6 |
| CHAPTER II STATE OF THE ART | 9 |
| 2.1 Classification of Existing Approaches | 9 |
| 2.2 Related Problems | 11 |
| 2.3 State of the Art | 13 |
| CHAPTER III PROBLEM DEFINITION | 18 |
| CHAPTER IV DISTRIBUTED AGENT PLACEMENT APPROACH | 21 |
| 4.1 Approach Overview | 21 |
| 4.2 Problem Reformulation | 24 |
| 4.2.1 Reformulation for Type-I Urban Zones | 24 |
| 4.2.2 Reformulation for Type-II Urban Zones | 26 |
| CHAPTER V AGENT PLACEMENT FOR TYPE-I URBAN ZONES | 31 |
| 5.1 Non-Model-Based Algorithms | 31 |
| 5.1.1 Main Assumptions | 31 |
| 5.1.2 Algorithm Description | 32 |
| 5.1.3 Algorithm Complexity | 34 |

| | | |
|---|--|-----------|
| 5.2 | Model-Based Algorithms | 36 |
| 5.2.1 | Target Motion Model | 36 |
| 5.2.2 | Model-Based Algorithm Description | 40 |
| 5.3 | Online Model Building | 42 |
| 5.3.1 | Order Estimation | 43 |
| 5.3.2 | Parameter Estimation | 45 |
| CHAPTER VI AGENT PLACEMENT FOR TYPE-II URBAN ZONES | | 48 |
| 6.1 | Non-Model-Based Algorithms | 48 |
| 6.1.1 | Assumptions and Background | 48 |
| 6.1.2 | Algorithm Description | 50 |
| 6.1.3 | Algorithm Complexity | 52 |
| 6.2 | Model-Based Algorithms | 54 |
| 6.3 | Optimizing Power Consumption | 55 |
| 6.4 | Dealing with Evasive Targets | 57 |
| 6.4.1 | Evasion Model | 58 |
| 6.4.2 | Dynamic Programming Approach | 60 |
| CHAPTER VII PERFORMANCE EVALUATION | | 66 |
| 7.1 | Approach Evaluation for Type-I Urban Zones | 66 |
| 7.1.1 | Non-Model-Based Algorithms: CEP | 66 |
| 7.1.2 | Model-Based Algorithms: PCEP | 70 |
| 7.1.3 | Online Model Estimation | 71 |
| 7.2 | Approach Evaluation for Type-II Urban Zones | 77 |
| 7.2.1 | Overall Coverage: Distributed Greedy Algorithms | 77 |
| 7.2.2 | Navigation Power Optimization: Auction Algorithm | 80 |
| 7.2.3 | Dealing with Evasive Targets: RTDP Approach | 80 |
| CHAPTER VIII CONCLUSION | | 85 |
| REFERENCES | | 87 |
| VITA | | 91 |

LIST OF FIGURES

| | | |
|-----------|--|----|
| Figure 1 | Hierarchical decomposition of a military surveillance mission. | 4 |
| Figure 2 | Classification of urban terrain zones. | 6 |
| Figure 3 | Two example UTZs. | 7 |
| Figure 4 | The W-CMOMMT problem. | 18 |
| Figure 5 | An overview of the proposed distributed agent placement approach. . . . | 22 |
| Figure 6 | Two example UTZs (repeated). | 25 |
| Figure 7 | Grid-type region of interest. | 26 |
| Figure 8 | A possible view from a mini-UAV hovering over a street intersection. . . | 27 |
| Figure 9 | A graph representing Type-II urban terrain zone. | 28 |
| Figure 10 | CEP pseudo-code representation. | 35 |
| Figure 11 | Predicting target location using a high-order Markov chain. | 40 |
| Figure 12 | Model-based agent placement using the probable cell outcomes. | 43 |
| Figure 13 | Overview of the model building approach. | 44 |
| Figure 14 | Estimating model order from partial autocorrelation plot. | 45 |
| Figure 15 | Modified Berchtold’s algorithm for estimating a transition matrix. | 46 |
| Figure 16 | Dividing the set of graph vertices among agents. | 51 |
| Figure 17 | Pseudo-code representation of the distributed greedy algorithm. | 52 |
| Figure 18 | Target transitions defined on a graph. | 54 |
| Figure 19 | Pseudo-code representation of a single iteration of the auction algorithm. . | 58 |
| Figure 20 | Target evasion model. | 59 |
| Figure 21 | System state represented by quantized probable edge outcomes. | 62 |
| Figure 22 | Pseudo-code representation of the RTDP algorithm. | 64 |
| Figure 23 | CEP vs. A-CMOMMT when targets are of the same utility value. | 68 |
| Figure 24 | CEP vs. A-CMOMMT when targets are of unequal utility values. | 69 |
| Figure 25 | A virtual $1\text{ km} \times 1\text{ km}$ urban region. | 70 |
| Figure 26 | PCEP performance compared to CEP as the target-to-agent ratio increases. . | 71 |
| Figure 27 | Estimating model order from the absolute value of the PACF | 73 |
| Figure 28 | Error in prediction due to using the estimated model. | 74 |
| Figure 29 | PCEP performance under actual and estimated models. | 75 |

| | | |
|-----------|--|----|
| Figure 30 | Estimation data size effect on the estimated model quality. | 76 |
| Figure 31 | A graph representing Type-II urban terrain zone. | 78 |
| Figure 32 | Performance of different placement algorithms for Type-II urban zones. . | 79 |
| Figure 33 | Optimizing average agent speed using the distributed auction algorithm. | 81 |
| Figure 34 | Performance of the RTDP algorithm. | 83 |
| Figure 35 | A case where all evasive targets are always observed. | 84 |

LIST OF SYMBOLS AND ABBREVIATIONS

Symbols

| | |
|------------------------|---|
| \mathcal{A} | Action space of a Markov decision process |
| $adj_{\mathcal{U}}(e)$ | A functions to determine if an edge e is adjacent to a vertex set \mathcal{U} |
| \mathbf{C} | Cost matrix |
| D_k | Maximum distance agent r_k can travel in a single sampling period |
| \mathcal{E} | A set of graph edges (street segments) |
| \mathcal{G} | Graph representation of the region of interest |
| H | Future time horizon |
| L | Solution set (used by the distributed greedy algorithm) |
| M | Order of a high-order Markov chain |
| m | Number of agents |
| N_x, N_y | Dimensions of a grid-type region of interest |
| n | Number of targets |
| O | Set of targets or algorithm complexity, depending on the context |
| o_j | A target |
| $P(\cdot)$ | Probability expression |
| \mathbf{Q} | Transition matrix |
| q_{ij} | Transition probability |
| R | Set of agents |
| r_i | An agent |
| \mathcal{R} | Reward function of a Markov decision process |
| \mathbb{R} | The set of real numbers |
| S | Sensing range of agent r_k |
| \mathcal{S} | Region of interest |

| | |
|----------------------|--|
| T | Tracking mission length |
| t | Time |
| \mathcal{T} | Transition function of a Markov decision process |
| \mathbf{U} | A vector representing target utility values |
| u_j | The j^{th} element in \mathbf{U} ; the utility value of j^{th} target |
| V | Value function |
| \mathcal{V} | A set of graph vertices (observation points) |
| \mathbf{X}_j | Vector representing probability distribution of target o_j location |
| (x_{sub}, y_{sub}) | A general location expression (defined in context) |
| \mathcal{X} | State space of a Markov decision process |
| χ_j | Location of target o_j |
| $\mathbf{\Gamma}$ | Boolean $m \times n$ matrix representing agent observations |
| γ_{ij} | An element in $\mathbf{\Gamma}$ (determines whether r_i is observing o_j) |
| Δ | Maximum vertex degree |
| Θ | Assignment set |
| λ_k | Contribution of lag k in an MTD representation of a high-order Markov chain |
| Φ | Set of exclusive observed reachable targets |
| Ψ | Set of locally observed targets |
| Ω | Set of observed reachable targets |

Abbreviations

| | |
|--------|---|
| AIC | <i>Akaike information criterion</i> |
| BIC | <i>Bayesian information criterion</i> |
| CEP | <i>Center-of-mass-based exclusive placement</i> |
| CMOMMT | <i>Cooperative multi-robot observation of multiple moving targets</i> |
| DP | <i>Dynamic programming</i> |

| | |
|----------|--|
| DSP | <i>Digital signal processing</i> |
| GMTD | <i>Generalized mixture transition distribution</i> |
| HC | <i>High-rise, closely spaced</i> |
| LP | <i>Linear programming</i> |
| LW | <i>Low-rise, widely spaced</i> |
| MDP | <i>Markov decision process</i> |
| MTD | <i>Mixture transition distribution</i> |
| NP | <i>Non-deterministic polynomial</i> |
| OP | <i>Observation point</i> |
| PACF | <i>Partial autocorrelation function</i> |
| PCEP | <i>Predictive center-of-mass-based exclusive placement</i> |
| ROI | <i>Region of interest</i> |
| RTDP | <i>Real-time dynamic programming</i> |
| SIMD | <i>Single-instruction-stream-multiple-data-streams</i> |
| SPMD | <i>Single-program-multiple-data-streams</i> |
| SR | <i>Sensing range</i> |
| UAV | <i>Unmanned aerial vehicle</i> |
| UGV | <i>Unmanned ground vehicle</i> |
| USAR | <i>Urban search and rescue</i> |
| UTZ | <i>Urban terrain zone</i> |
| W-CMOMMT | <i>Weighted cooperative multi-robot observation of multiple moving targets</i> |

SUMMARY

The problem of dynamic autonomous agent placement for tracking moving targets arises in many real-life applications from the civil as well as the military domains, such as rescue operations, security, surveillance, and reconnaissance. The objective of this doctoral thesis is to develop a distributed hierarchical approach to address this problem. After the approach is developed, it is tested on a number of urban surveillance scenarios.

This thesis classifies existing approaches according to a number of independent criteria. The work presented here belongs to the most general class based on each classification criterion. Some of the current state of the art approaches assume some knowledge of target behaviors in the form of a model while others are non-model-based. A number of them couple the agent placement with motion planning, and others employ a game-theoretic framework to reason about agent placement for tracking non-cooperating targets. None of the available methods integrates non-model-based and model-based tools along with an online model building process, as suggested in this thesis.

The proposed distributed approach views the placement problem as a multi-tiered architecture entailing modules for low-level sensor data preprocessing and fusion, decentralized decision support, knowledge building, and centralized decision support. This thesis focuses upon the modules of decentralized decision support and knowledge building. The decentralized decision support module requires a great deal of coordination among agents to achieve the mission objectives. The module entails two classes of distributed algorithms: non-model-based algorithms and model-based algorithms. The first class is used when no target motion model is available. Thus, it is used as a place holder while a model is built to describe agents' knowledge about target behaviors. After the model is built and evaluated, agents switch to the model-based algorithms to make better placement decisions using the model.

To apply the approach to urban environments, urban terrain zones are classified, and

the problem is mathematically formulated for two different types of urban terrain, namely low-rise, widely spaced and high-rise, closely spaced zones. An instance of each class of algorithms is developed for each of the two types of urban terrain. The algorithms are designed to run in a distributed fashion to address scalability and fault tolerance issues. The class of model-based algorithms includes a distributed model-based algorithm for dealing with evasive targets. The algorithm is designed to improve its performance over time as it learns from past experience how to deal with evasive targets. Apart from the algorithms, a model estimation module is developed to build motion models online from sensor observations.

The approach is evaluated through a set of simulation experiments inspired from real-life scenarios. Experimental results reveal the superiority of the developed algorithms over existing ones and the applicability of the online model-building method. Therefore, it is concluded that the overall distributed approach is capable of handling agent placement for surveillance applications in urban environments among other applications.

Thus, the main contributions of this doctoral thesis can be summarized as follows:

- a distributed hierarchical approach to the agent placement problem that is applicable to a wide variety of target tracking applications.
- a study of how to apply the approach to low-rise, widely spaced and high-rise, closely spaced urban terrain zones.
- a parsimonious target motion model that captures both the random nature and the dynamics of target behaviors.
- a number of distributed agent placement algorithms for the two types of urban terrain.
- an intelligent algorithm that learns how to deal with evasive targets.
- an online model estimation methodology to learn the target motion model from sensor observations.

CHAPTER I

INTRODUCTION

This chapter discusses the motivation for studying the problem of interest to this doctoral thesis and provides some background material. Then, it shows how the rest of the thesis is organized.

1.1 Motivation

An important issue that arises in the automation of many security, surveillance, and reconnaissance tasks is that of observing the movements of targets navigating in a bounded area of interest [31] [33]. Automation is necessary for two main reasons. The first main reason is that, as the complexity of the problem (in terms of the number of targets, the degree of target evasiveness, the size of the region of interest, etc.) increases, placing/replacing agents¹ in an ad-hoc manner is highly expected to result in poor coverage. In such situations, more sophisticated methods are to be used to achieve acceptable coverage. The second main reason is that, in dangerous missions, it is desirable to keep the human factor out of the picture to avoid the loss of human life. For example, monitoring a hazardous spot following a nuclear disaster is better done with unmanned agents so as to avoid the risk of exposing human beings to harmful radiations. Another example is military reconnaissance over hostility regions where it is highly probable to lose one or more reconnaissance agents during the mission; using unmanned agents again is a way to keep human lives out of harm's way.

The size of the region of interest, along with the sensing range of agents, plays an important role in choosing the proper framework for the problem of interest. For small regions, a single agent with an appropriate sensing range may be sufficient to observe all

¹Throughout this thesis, the term “agent” is used to refer to a sensor that possesses some motion, processing, and communication capabilities, such as a robot, an unmanned aerial/ground vehicle, etc.

of the moving targets. If the size of the region of interest increases, however, multiple agents will be required to achieve reasonable coverage, especially since agents may only have limited sensing ranges. A key issue that arises in this case is how to place the agents over the region of interest. Fixed deployment can arguably be used in situations where the region of interest is small enough that the number of available agents can achieve the desired coverage or in situations where there is no limit on the number of agents to be used. For economic reasons, however, the number of available agents may be limited, rendering the fixed agent deployment approach infeasible for the problem under consideration. In such cases, dynamic placement of a limited number of agents becomes an attractive option. When sensors are mounted on mobile agents, the (dynamic coverage) problem is reduced to one of agent positioning. Such a capability is of obvious use in the detection of unfriendly targets (e.g., military operations), monitoring (e.g., security), or urban search and rescue (USAR) in the aftermath of a natural or man-made disaster (e.g., building rubble due to an earthquake or other causes) [7].

1.2 Agent Placement versus Target Tracking

The term *target tracking* is often used to refer to the task of finding/estimating the motion parameters (mainly the location and direction) of a certain moving object in a time sequence of measurements. This task is achievable as long as the target is within the sensor's sensing range. For example, in computer vision, target tracking can be defined as the task of finding a moving object in a sequence of images. If it happens that the object keeps moving away to the point it runs off the field of view of the camera, then the target tracking task will fail to track the moving object until the object re-enters the camera's field of view. To address such problems, one can mount the camera on a moving platform such as a robot or an unmanned vehicle. We call the new setup (the camera plus the robot or the unmanned vehicle) an agent. Thus, we can start a second task, other than the target tracking task, to move the camera accordingly to guarantee that the object stays in view. In more general terms, the second task is responsible for (reactively or proactively) moving the agent to guarantee that the moving object stays within the sensor's sensing range. That second task

is what we call the *agent placement* task. The so-called *active sensing-based target tracking* forms the glue between the two tasks.

As we can see, the two terms, *target tracking* and *agent placement*, are closely related to the point that some researchers may use them interchangeably. Nevertheless, this thesis draws the distinction between the two tasks beforehand since the work presented in the following chapters is primarily concerned with the task of agent placement. The task of target tracking (as defined above) is beyond the scope of this thesis. Therefore, by using the term *agent placement for target tracking*, we primarily refer to the *agent placement* task that may be coupled with an existing *target tracking* task.

1.3 Surveillance for Urban Warfare

Although the approach proposed in this thesis is general enough that it can be coupled with a wide variety of target tracking systems, surveillance applications emerging in urban warfare is used as an example to evaluate the approach. The following subsections first address where the work presented in this thesis may fit in the context of military operations. Then, the main difficulties encountered in monitoring urban environments are briefly discussed.

1.3.1 Hierarchical Decomposition of Surveillance Missions

This thesis views a surveillance mission as a multi-tiered architecture entailing modules of situation awareness, optimum (or near-optimum) agent deployment for maximum coverage, and dynamic agent placement for observing moving targets (see Figure 1 for a schematic representation). The agents are charged with the responsibility of conducting target surveillance and reconnaissance by placing themselves optimally. They are capable of identifying and tracking moving targets. Target classification and prioritization allow for critical targets to be continually observed even with limited available resources.

The module of situation awareness is concerned with making use of the available information to develop a comprehensive view of “what is going on” in an environment (especially in a battle space). The module of situation awareness is used to make high-level decisions based on prior knowledge, current data, and future predictions. When a decision is made to perform a certain mission, relevant information is passed to the mission modules from the

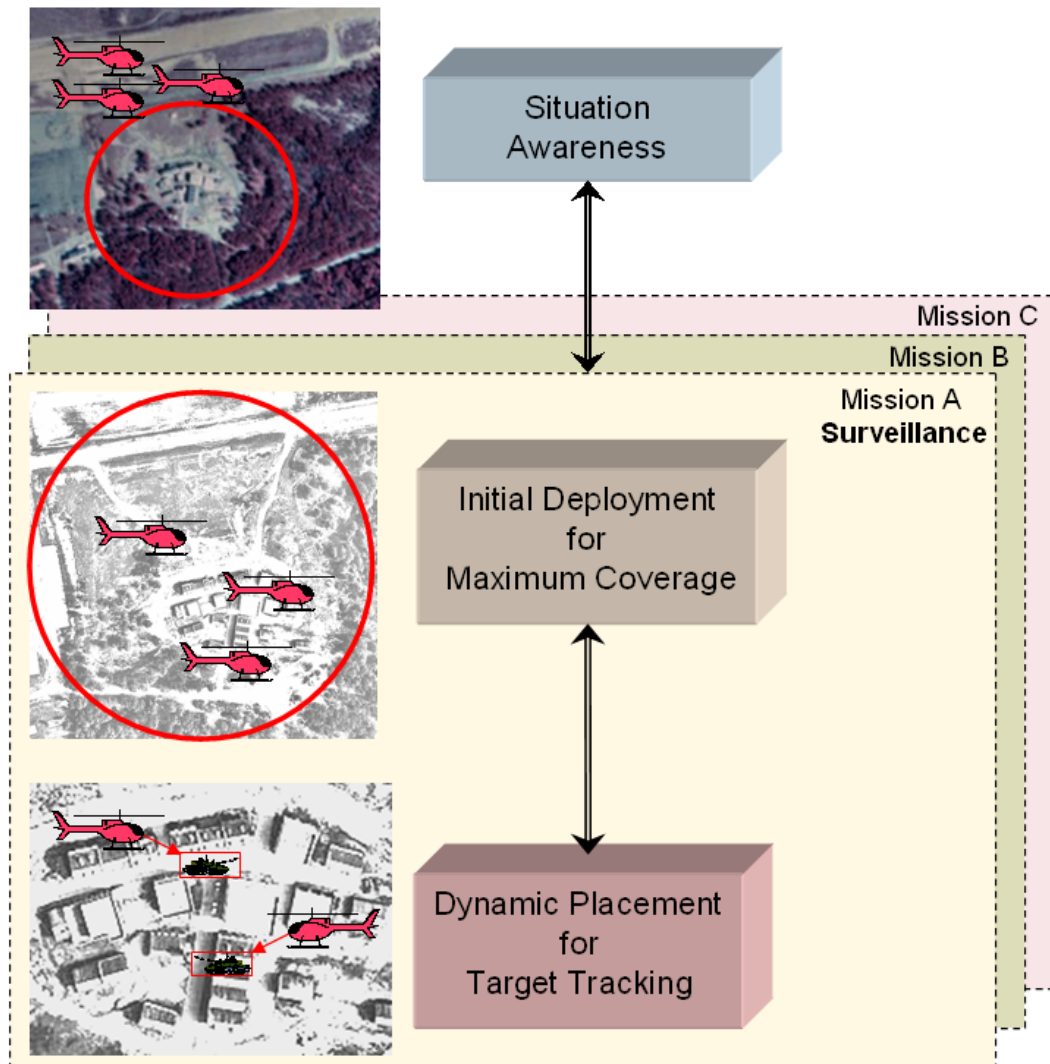


Figure 1: Hierarchical decomposition of a military surveillance mission.

situation awareness module. Thus, in the case of surveillance missions, information about routes, target priorities, possible current and future target locations, etc. is passed to the surveillance mission modules. For a surveillance mission, two tasks are to be performed as viewed by this thesis. The first task is concerned with initial agent deployment for maximum coverage based on the information available from the situation awareness module, including intelligence information about current and future target locations. In the absence of such information, an alternative exploration task would be required to seek out targets, which is outside the scope of this thesis; initial agent deployment is done in light of the information provided by the situation awareness module. The second task is concerned with dynamic agent re-planning based on the changes in target locations.

1.3.2 Surveillance in Urban Zones

According to a variety of hypotheses that have been posited, urban military operations will become more frequent. Urban surveillance and reconnaissance tasks are more demanding than those on most other types of terrain. This is mainly due to poor line of sight and intense clutter, which make some sensor platforms of much less value in urban environments [38]. The line of sight problem arises as tall buildings tend to partially or fully occlude vision of city roads, which may contain important targets. As a result, this can limit the effective sensing range significantly. Ellefsen has developed an Urban Terrain Zone (UTZ) classification system based on the physical characteristics and spatial patterns in different parts of cities. Based on a detailed study of several cities from North America, Europe, the Middle East, and Asia, Ellefsen [17] specified seven UTZ classes. We can roughly collapse the seven classes into four classes (see Figure 2):

- high-rise, closely spaced,
- high-rise, widely spaced,
- low-rise, closely spaced, and
- low-rise, widely spaced.

Each class usually represents certain parts of the city, e.g., residential areas, business areas, etc. Figure 3 shows examples of two different UTZ categories.

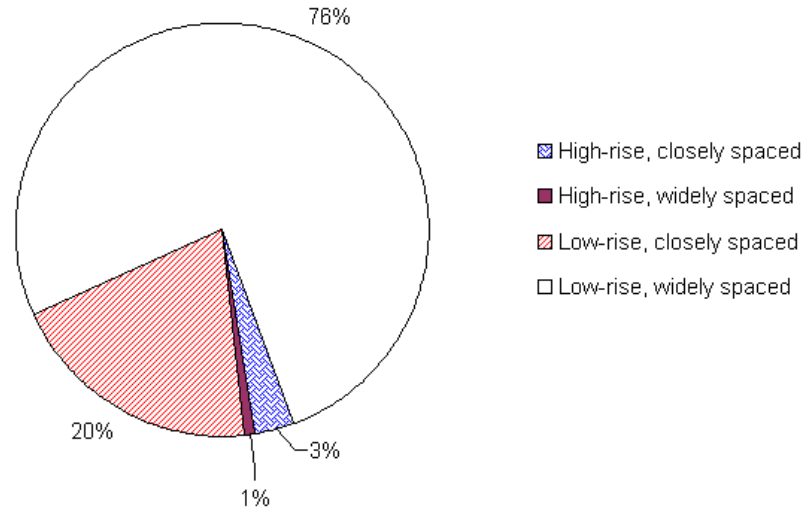


Figure 2: Classification of urban terrain zones.

When performing aerial surveillance on urban streets, line of sight occlusion does not cause a significant problem for areas occupied by widely spaced low buildings, which typically form about 76% of the city (Figure 2). Therefore, the surveillance problem can be treated as a two-dimensional problem using aerial down-looking agents flying at a reasonably high altitude. In this case, buildings are expected to cause slight or zero occlusion. However, the small percentage of the city area that contains high-rise buildings is indeed of a great significance since those buildings are expected to contain most of the important cultural, economic, and administrative structures of the city [38]. Performing aerial street surveillance on such areas is not an easy task. Monitoring the vertical surfaces of the buildings in addition to the streets may require employing several kinds of agents such as mini-unmanned aerial vehicles (mini-UAVs) and unmanned ground vehicles (UGVs), in addition to high altitude endurance UAVs.

1.4 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides a survey of the state of the art in solving the problem of dynamic agent/sensor placement for surveillance applications



(a) Low-rise, widely spaced UTZ.



(b) High-rise, closely spaced UTZ.

Figure 3: Two example UTZs.

and related problems. Chapter 3 provides the general statement of the dynamic agent placement problem. In Chapter 4, the proposed distributed approach is outlined, and the problem of interest is reformulated for surveillance applications in two different types of urban terrain zones: low-rise, widely spaced and high-rise, closely spaced zones. The following two chapters (Chapter 5 and Chapter 6) discuss the application of the approach to each type. The chapters provide a number of algorithms to address each case. Chapter 7 contains the simulation-based experimental setup and the performance evaluation results. Finally, Chapter 8 provides the conclusions drawn from the research conducted throughout this doctoral thesis.

CHAPTER II

STATE OF THE ART

This chapter is to mainly provide a survey of existing approaches to the agent placement problem for tracking moving targets. The chapter starts by introducing a number of classifications for the existing approaches, discussing each class at an abstract level, and showing to which classes the proposed approach belongs. The classification is followed by a brief discussion of the most related problems to the problem of interest to this thesis since some of the existing approaches overlap with other problem areas. Finally, a survey of the most relevant works is presented.

2.1 Classification of Existing Approaches

Several approaches have been devised by researchers to address the problem of interest to this thesis. To better organize the survey and to show where the work presented in this thesis fits in the state of the art, we classify the existing approaches according to different independent criteria. Table 1 shows the taxonomy used to categorize existing approaches to the problem under consideration.

Approaches can be classified according to how many agents and targets each approach

Table 1: Classification of existing approaches to the agent placement problem.

| Classification Basis | Classes |
|------------------------------------|---|
| Multiplicity of agents and targets | Single-agent, single-target Single-agent, multi-target Multi-agent, multi-target Multi-agent, multi-target |
| Centrality | Centralized Distributed (decentralized) |
| Predictiveness | Non-predictive Predictive |

can handle. This criterion spawns four different categories (Table 1). The single-agent, single-target case is probably the simplest among the four categories. Adding more targets and agents is bound to increase the complexity of the problem by causing an exponential blow up in the size of the state space and the action space, respectively. This concept is widely known as the “curse of dimensionality” [9]. Alleviating the intractability associated with the curse of dimensionality in a distributed setup, where agents can communicate with one another, entails high degree of coordination among agents. The problem addressed in this thesis falls in the multi-agent, multi-target category, which is the hardest and the most general among the four categories.

Independently from the classification discussed in the previous paragraph, multi-agent approaches can be classified as either centralized or distributed. The known superiority of distributed approaches to centralized approaches is due to a number of reasons. When the size of the problem of interest increases, scalability issues get raised; some approaches are only meant for problems of a small size and would not work well for problems of larger sizes. In these cases, distributed algorithms are known to scale better than centralized ones. Agents can share the computational load and exploit parallelism to speed up the decision making process. In some cases, the severity of the scalability problem can reach a point where the centralized approach may fail to keep up with the size of the problem. A good example for this is the problem of interest to this thesis. Consider a scenario where it is required to perform a surveillance task over a huge metropolitan area that spans hundreds of square miles using tens of agents. For example, assume that the scenario involves 80 agents, each of which is loaded with only one type of sensors: a high-resolution vision camera. Assume further that each camera delivers twenty 640×480 color frames every second. For a strictly centralized approach where no processing (including data compression) is available onboard any agent, the central node would need to receive data at

$$80 \times \left(20 \frac{\text{frame}}{\text{second}}\right) \times \left(640 \times 480 \frac{\text{pixel}}{\text{frame}}\right) \times \left(3 \frac{\text{color}}{\text{pixel}}\right) \times \left(8 \frac{\text{bit}}{\text{color}}\right) \approx 11\text{Gbps}, \quad (1)$$

which is a considerably high bandwidth by today’s standards, let alone the tremendous computing power that would need to exist on a single processing node.

Scalability is not the only advantage of distributed approaches. Fault tolerance is another main concern especially in mission-critical systems. Centralized systems suffer from an inherent problem for that matter as they possess a single point of failure, which is the central agent where all the processing and decision making take place. Distributed systems, on the other hand, tackle this problem by eliminating this single point of failure and distributing the decision making responsibility among a number of agents. A robust distributed approach would recover from individual agent failures easily and quickly. Thus, the approach proposed in this thesis is a distributed approach for all the reasons described above.

A different way to classify the agent placement strategies is based on the approach's predictiveness. A predictive approach would use the current observed target locations, a history of target locations, and possibly a motion model to predict the future target locations with some degree of uncertainty, whereas a non-predictive approach would assume that the targets stay in their current observed locations at least until a short time before the next measurement is taken. The advantage of predictive methods is that they can act proactively to relocate the agents to account for target behaviors and increase the chance of achieving better coverage. However, the model used for prediction should hold a reasonable degree of accuracy, or else it may have a negative effect on the performance. The work presented in this thesis combines both predictive as well as non-predictive schemes, along with online model building, as will be discussed in detail in the following chapters.

2.2 Related Problems

The agent placement problem is related to a number of other problems such as the pursuit-evasion problem from game theory [22], the grid coverage problem [18], the motion planning problem [24], the art gallery problem [28], the vertex cover problem from graph theory [42], and others. The following few paragraphs discuss some of those problems and shed light on the salient distinctions between each problem and the problem under investigation in this thesis.

The agent placement problem addressed in this thesis can be considered as some form of

the pursuit-evasion problem, where pursuers correspond to agents, and evaders correspond to targets. Pursuit-evasion games are problems of fundamental interest in many fields such as computer science, operations research, game theory, and control theory. The goal of a pursuit-evasion game is to find a strategy for a pursuer trying to catch an evader, which, in turn, tries to avoid capture indefinitely [22]. This definition can obviously be extended to multi-pursuer, multi-evader cases. Most pursuit-evasion games are designed to terminate upon capturing the evader(s). An evader is said to be captured if a pursuer manages to exist in the same location as the evader up to some spatial resolution. In our version of the agent placement problem, on the other hand, no *capture* event is defined. Thus, agents are required to constantly monitor targets, and the mission is assumed to continue indefinitely. The goal of the agent placement problem then is to keep as many important targets as possible in the agents' field of view as long as possible.

Gage [18] divides the dynamic coverage problem into three groups of behaviors. First, *blanket coverage* aims to achieve a static arrangement of agents to maximize the detection rate of targets. Second, *barrier coverage* aims to achieve a static arrangement of agents with the task of minimizing the probability of undetected target penetration through the barrier (The barrier is formed by agents lined up according to a certain pattern). Finally, *sweep coverage* represents moving barrier coverage or can be achieved using random uncoordinated motion of agents as in [18]. According to Gage's taxonomy of the coverage problem, the problem of interest to this thesis falls under the blanket coverage category since the agents are spread out over the region of interest without necessarily forming a barrier. However, there is at least one fundamental difference between the solution proposed in this thesis and Gage's definition of blanket coverage. While blanket coverage aims to achieve a static arrangement to maximize target detection rate, our agent placement problem does not seek to achieve a static arrangement. The objective is to keep as many important targets as possible in view by moving agents whenever needed.

The basic motion planning problem deals with finding a path or a sequence of locations to guide a moving agent. The path is required to achieve some objective, such as avoiding obstacles, minimizing the distance traveled by an agent, etc. Accordingly, the motion

planning problem is a general problem that can address various application domains. If the planning goal is set to maximizing the time during which some target is kept under agent surveillance, the problem becomes of the same variety of problems considered in this thesis. La Valle *et al.* [25] have addressed such problem for the single-agent, single-target case.

2.3 State of the Art

This section provides a brief survey of the most relevant past efforts that have addressed the problem of dynamic agent placement for target tracking. We attempt to classify each approach based on the taxonomy presented in Section 2.1. The relevance of each approach to the problem under consideration is identified. Finally, pros and cons of each approach are pointed out.

Parker *et al.* [31] [32] [34] have developed a number of approaches to the CMOMMT [31] problem, which is one of the formally defined versions of the problem of interest to this thesis. The approaches fall into the categories of multi-agent, multi-target, distributed, and (mostly) non-predictive. Parker presented a hand-generated heuristic solution called A-CMOMMT [31], in which local force vectors are used to direct moving agents with limited sensing range in order to minimize the time targets escape surveillance by at least one agent. The local force vectors are designed to attract an agent to the targets and repel them away from other agents. The main advantages of this approach include simplicity and scalability. It is also referred to as the “best-known human-designed policy” for the CMOMMT problem [37]. Then, Parker introduced a lazy Q-learning approach [32] [34] that contributed two main conclusions to the same problem. First, a dynamic deployment policy with learning was confirmed to outperform a random policy, and more interestingly, some user defined-policies. Second, because the proposed Q-learning approach did not take into account the neighboring agents, a user-defined policy, such as A-CMOMMT, which takes this information into account, is bound to exhibit better performance than the proposed Q-learning approach [34]. In [31], the author combines the low-level local force vectors approach with the higher-level information (the probability that a certain target actually exists and the probability that no other agent is already observing a given target) using

her previous effort, ALLIANCE [29] [30], to achieve better overall performance. Except for some of the work presented in [31], the contributed approaches do not consider the high-level information that the user may already have (e.g., intelligence information in a military application). Including high-level information in designing an agent placement algorithm is expected to significantly improve the algorithm performance. The collection of approaches contributed by Parker *et al.* does not consider a target motion model for predicting future target locations.

Vidal *et al.* [39] presented a probabilistic approach to pursuit-evasion games involving UAVs and UGVs. They considered two computationally feasible greedy pursuit policies: *local-max* and *global-max*. The pursuers chase the evaders while building a map in an unknown environment. Since the authors considered a pursuit-evasion game, expected capture time was used as the primary performance metric. The policies were implemented on real pursuers and evaders. The work falls in the predictive, multi-agent, multi-target, and distributed categories. The strength of the work is mainly due to the distributed nature of the approach and the capability of predicting future target locations. The work presented in this thesis mainly differs from Vidal’s in one aspect. As pointed out above, in this thesis, the main objective is to keep targets under surveillance for as long as possible. No *capture* event is defined, and it is assumed that the mission is not interested in capturing the targets. Therefore, the game-theoretic framework used by Vidal would not be applicable in this case.

Batalin and Sukhatme [8] [7] have contributed a number of approaches to the dynamic coverage problem. The approaches are classified as distributed, multi-agent, multi-target, non-predictive approaches. In [8], they propose deploying a mobile sensor network by dispersion and then applying local rules for sensor coverage maximization. The premise of this algorithm is that in order to achieve good coverage as a team, agents must “spread out” over the environment. If agents are to be placed too close to one another, their fields of view will overlap, which will result in poor overall coverage. It is clear that this approach resembles Parker’s A-CMOMMT, which uses local force vectors and thus has the same main advantages and disadvantages as A-CMOMMT (i.e., it is simple and scalable but does not take advantage of higher-level knowledge). Batalin and Sukhatme’s other

approaches [7], on the other hand, assume that the coverage requirements are such that every point in the environment should be covered with a certain frequency. The approaches deploy static and mobile sensor networks and produce exploratory, patrol-like behavior. Agents deploy communication beacons into the environment to mark previously visited areas. These nodes act as local signposts for agents, which subsequently return to their vicinity. By deploying such (stationary) nodes into the environment, agents can make local decisions about their motion strategy. The main advantage of these approaches is that they are designed under the fundamental constraint that global knowledge about the environment (e.g., Global Positioning System or a map) is unavailable. In addition, the algorithms are fully decentralized, making them fault tolerant. However, one can argue that incorporating the constraint of global knowledge unavailability may not always be considered of high significance, especially in military applications where agents are usually equipped with a Global Positioning System device. As we can also see, the authors did not consider a motion model for the targets, which will be shown later in this thesis to lead to better performance.

The author in [43] discusses the problem of deployment of distributed sensors in the wireless ad hoc network domain mainly for data collection. The scheme can be classified as a distributed, multi-agent, multi-target scheme. It may be classified as predictive although the prediction technique used is simple: a random walk algorithm. The communication ranges between the robots are assumed to be limited, and the environment is large enough so that the network connectivity cannot be maintained at all times. A random-walk algorithm is used to disperse the robot network into the environment to support communication. The work reveals that, in mission scenarios that do not require data collection in real time and where occasional data loss is acceptable, the proposed data collection mechanism is feasible, robust, and economical in terms of the number of robots required. Although there is an overlap between the work presented in [43] and the work introduced in this thesis, the focus of the former is on maintaining communication paths between agents, not on maximizing the coverage of targets over time. The conclusion of the work indicates that the feasibility of the approach is questionable when considering a real-time application such as the one under consideration in this thesis.

La Valle *et al.* [25] introduced motion strategies for maintaining the visibility of moving targets. The strategies can be classified as centralized, single-agent, single-target, and predictive. The sensing agent here is a robot with mounted vision cameras. Target motion is assumed to be fully or partially predictable. For the fully predictable targets, the authors presented an algorithm that computes optimal numerical solutions using a dynamic-programming framework. For the partially predictable targets, the authors presented two online algorithms. The first algorithm adopts the strategy of maximizing the probability that the target will remain in view in the subsequent time step while the other algorithm adopts the strategy of maximizing the minimum time in which the target could escape the visibility region. The main advantage of the approach taken in La Valle’s work is that it considers a motion model for the target, which is used for prediction. However, the work seems to be limited to the single-agent, single-target scenario. The complexity of the problem is bound to increase significantly when considering multi-observer, multi-target cases, especially if the problem entails cooperation and coordination between observers to achieve the global objective of the mission.

Cook *et al.* [14] [15] used a decision-theoretic approach to optimal agent placement, which can be categorized as a distributed, multi-agent, multi-target, predictive approach. Static and dynamic planning are done using multi-attribute utilities and decision theories. The main goal is to maximize the value of information gained by sensors. Cooperation between agents is used to balance the workload and increase the information gain. Sensor planning is performed based on the trade-off between two conflicting attributes: maximizing the value of information acquired and minimizing the exposure to the enemy. The observation point refinement algorithm generates a sorted list of observation point sets (observation point for each agent), where each set is rated by the utility measures. To choose the selection points, the algorithm performs coarse sampling to the search space (which could be an area, a path, etc.) for each agent. Then, “optimal” solutions are sought in the vicinity of the most promising sampling points. Although the authors call their solution an optimal solution, it is believed to be only suboptimal, since there is no guarantee for the optimality of the decisions. The strength behind Cook’s approach is that it assumes some

knowledge about target navigation with some degree of uncertainty. Among the drawbacks of Cook's approach, as understood from the articles [14] [15], is the expensive computational complexity associated with calculating utility factors for several agents over a large area of interest.

Isler *et al.* [21] studied the problem of capturing a single evader using one or more pursuers moving on a graph. At each round, the evader tries to gather information about the location of the pursuers but it can see them only if they are located on adjacent nodes. It is shown that two pursuers suffice for catching an evader with such local visibility with high probability. The authors distinguish between reactive evaders that move only when a pursuer is visible and general evaders that can employ more sophisticated strategies. The work presents polynomial time algorithms that decide whether a graph is pursuer-win, that is, if a single pursuer can capture an evader of either kind on the graph. The work belongs to the multi-agent, single-target, predictive category. Isler's work differs from the work presented in this thesis in two main aspects. First, this thesis considers multi-target scenarios, whereas Isler's work mainly addresses single-target cases. Second, Isler's approach is developed in a game-theoretic framework, where there is a capture event that ends the game while this is not the case in the problem considered in this thesis.

CHAPTER III

PROBLEM DEFINITION

The problem of dynamic agent placement for target tracking can be stated as:

Given a set of moving targets with different utility values and a set of agents with limited sensing range in a bounded enclosed spatial region, how can the agents be dynamically placed to maximize the time that important targets are under the surveillance of at least one agent?

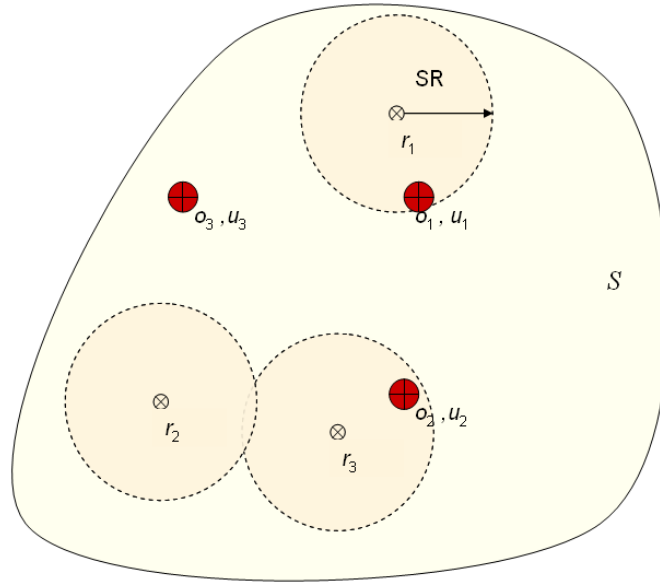


Figure 4: The W-CMOMMT problem.

There exists a number of efforts to formally describe the dynamic agent placement problem for target tracking. The choice is made to use a formulation of the variety of *Weighted Cooperative Multi-robot Observation of Multiple Moving Targets* (W-CMOMMT) [40] [41] since it captures the multiple-observer-multiple-target scenario with target prioritization (see Figure 4). W-CMOMMT defines the dynamic agent placement problem for tracking

multiple moving targets as follows:

Given:

- \mathcal{S} : a bounded, enclosed spatial region,
- $R = \{r_i | i = 1, 2, \dots, m\}$: a team of m agents with observation sensors that are noisy and of limited sensing range (SR),
- $O(t) = \{o_j(t) | j = 1, 2, \dots, n\}$: a set of n targets, such that target $o_j(t)$ is located within region \mathcal{S} at time t , and
- \mathbf{U} : a vector of utility values such that u_j reflects the utility value gained by observing target o_j .

Define an $m \times n$ Boolean matrix $\Gamma(t)$, where

$$\gamma_{ij}(t) = \begin{cases} 1 & \text{if } o_j \text{ is detectable by } r_i \\ 0 & \text{otherwise} \end{cases}. \quad (2)$$

The goal is to provide a dynamic agent placement policy to maximize the normalized global utility function

$$G = \frac{\int_0^T \sum_{j=1}^n u_j \bigvee_{i=1}^m \gamma_{ij}(\tau) d\tau}{T \sum_{j=1}^n u_j}, \quad (3)$$

where \bigvee is the logical disjunction operator, and T is the time interval during which the tracking mission is carried out.

It is noteworthy that CMOMMT [31] is a simpler version of our problem, W-CMOMMT, since the former assumes that targets are equally important. CMOMMT is an NP-hard problem that requires strong cooperation for good performance [40] [41] [33]. Thus, we conclude that the problem under consideration in this thesis, which is a harder problem than CMOMMT, is NP-hard as well.

The problem as defined above is believed to be general enough to describe a wide variety of scenarios. Several customizations can be made to adapt the problem to the scenario under study. For example, throughout this thesis, discrete time is used since most smart sensors that can be mounted on an agent deliver sampled signals to the agent. For instance, a

digital video camera delivers image frames at a certain rate. The spatial region of interest \mathcal{S} may also be continuous or discrete. Finally, we can add a set of static or dynamic obstacles that occupy some of the region \mathcal{S} . Including such obstacles increases the difficulty of the problem along two dimensions. First, agents cannot move freely in \mathcal{S} as they have to avoid the obstacles. Second, obstacles may come in the way between an agent and a target, limiting the former's field of view. An intelligent target may also use the obstacle to hide from the agent. As will be shown in the following chapters, a number of customizations are made to the problem definition depending on the case to be addressed.

CHAPTER IV

DISTRIBUTED AGENT PLACEMENT APPROACH

This chapter begins by providing an overview of the proposed distributed approach for the agent placement problem and defining the scope of the research addressed in this doctoral thesis. The general problem defined in Chapter 3 is then reformulated to fit the scope of this thesis.

4.1 Approach Overview

An overview of the proposed hierarchical approach is depicted in Figure 5. The dashed box represents the functionality of one agent. An agent communicates with peer agents as well as with a common command and control module. In the following few paragraphs, the function of each module is explained.

At the lowest level, an agent processes sensor data and locally fuses data collected from different local sensors. For example, a UAV that has a vision camera and an infrared camera onboard needs to preprocess both kinds of images separately in real time to detect and identify different targets. In addition, the agent may need to fuse vision images and infrared images to reduce the uncertainty associated with the target detection/identification task. Recent advances in many research areas including computer vision (e.g., [13] and [26]) have made it possible to achieve those tasks in a timely manner. Therefore, the target identification, classification, and tracking problems are outside the scope of this thesis; it is assumed that an agent is equipped with real-time target identification and classification software.

Intelligent decision support is provided at two different levels. First, decentralized decision support is performed at the agent level where cooperation and coordination among agents are necessary to achieve the mission objective. Second, centralized decision support is provided at the command and control level, where all necessary information is collected

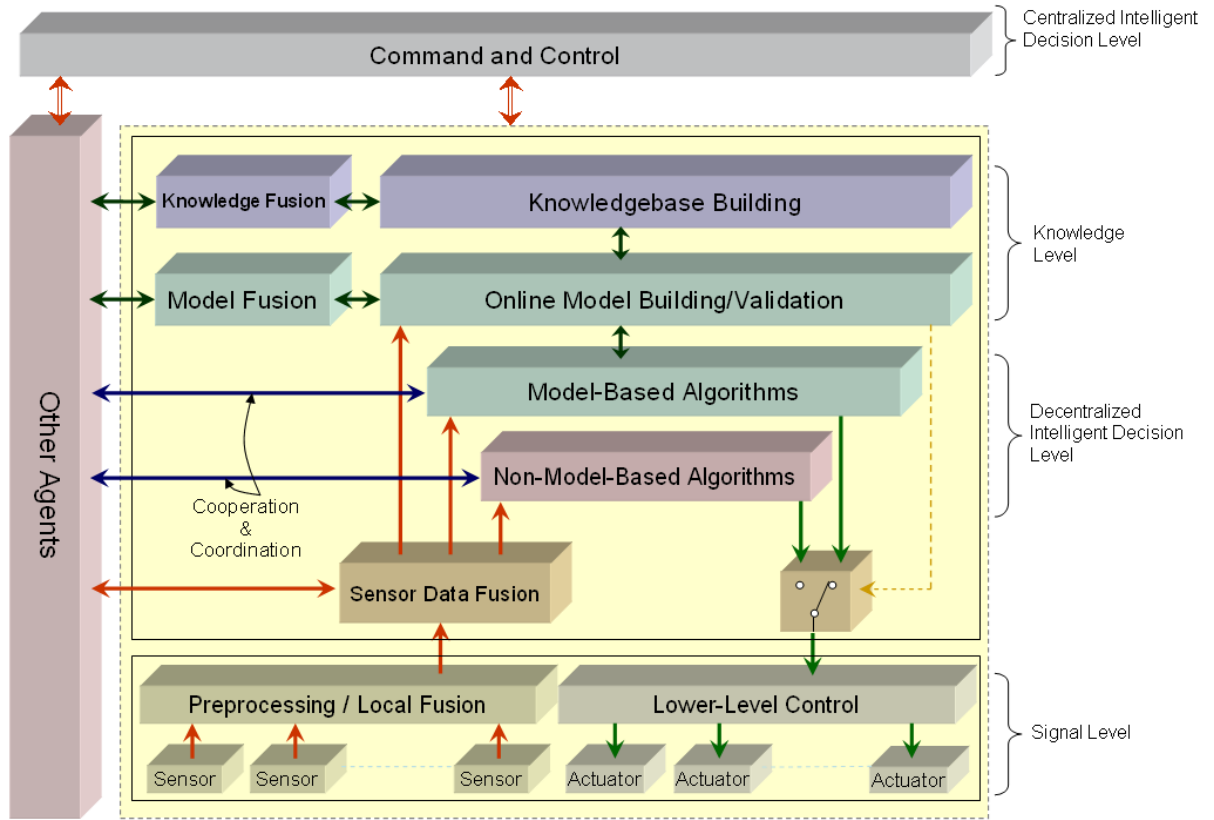


Figure 5: An overview of the proposed distributed agent placement approach.

and processed. Collecting and processing information at a central command and control location can be a very costly process in terms of communication bandwidth, processing power, and real-time requirements. As a result, centralized decisions are to be made much less frequently than decentralized decisions. For example, it may be necessary to make a centralized decision when a commander finds it necessary to switch the focus of the whole mission to a different region, whereas a decentralized decision is to be made when an agent detects that a certain target has traveled half a mile east. The proposed decentralized decision support adopts a combination of both non-model-based and model-based algorithms. Model-based algorithms use stochastic motion models to represent agents' knowledge about the behavior of each type of targets. When a target motion model is not available, non-model-based algorithms are used as place holders, while a cooperative online model learning task takes place. After a model has been built and evaluated, agents can switch to using

model-based algorithms.

At a higher level comes the motion model and the knowledgebase to represent the agent's knowledge about the target behaviors and about the best policies that the agent has tried in the past. Each agent builds a model using fused sensor observations. Agents share their knowledge about target behavior by fusing the target models they have built separately.

Agents communicate at different levels of abstraction to achieve several goals. First, they communicate at the signal level in order to fuse their local sensor information. For example, in a situation where each agent has detected the same target to a certain degree or certainty, agents can fuse their information to reduce the uncertainty bounds associated with the detection task. Second, agents communicate at the decentralized decision support level in order to cooperate and coordinate their efforts to better achieve the global objective. Consider a situation where two or more agents have detected a highly important target. If agents were to optimize their local goals only, each of them would follow that important target and possibly leave out other less important targets. In this case, each agent would have optimized its local utility at the expense of the mission's global utility, which would have been optimized if one agent had monitored that important target while other agents monitored other targets. Agent cooperation and coordination is a crucial issue in distributed agent placement. Finally, agents communicate at the knowledge level in order to share their knowledge about target behaviors and their experience of dealing with different mission scenarios.

The output of the decentralized decision level is the next best observation points for each agent. To actually move an agent to a new observation point, a trajectory from the current observation point to the next observation point needs to be generated. The trajectory is required to avoid possible collision with targets, other agents, and obstacles. The trajectory generation task is carried out by the low-level control module. This module is outside the scope of this thesis.

4.2 Problem Reformulation

Referring to Section 1.3.2, this thesis divides the problem of monitoring urban terrain zones (UTZs) into two main categories: monitoring low-rise, widely separated (LW) and monitoring high-rise, closely separated (HC) zones. For simplicity, LW zones and HC zones will sometimes be referred to as Type-I and Type-II urban terrain zones, respectively. Figure 6(a) shows an example of LW zones¹. In such kind of environment, the agent placement task can be performed by employing aerial agents flying at a reasonable altitude and looking down at the region of interest. It is clear that all the streets can be seen with no major occlusion. HC zones, on the other hand, cause a severe line-of-sight problem (see Section 1.3.2) as clearly seen in Figure 6(b). The following two subsections reformulate the problem for the two different types of UTZs.

4.2.1 Reformulation for Type-I Urban Zones

As described in Section 4.1, the proposed approach uses two different classes of algorithms. We will use a different reformulation for each class. The formulation for the non-model-based class is the same as the one presented originally in Chapter 3 with the only exception that discrete time is used instead of continuous time. The reason is that most smart sensors that can exist onboard a surveillance agent already deliver sampled data (e.g., a digital video camera). Sampling makes it easier to preprocess the sensor data using DSP chips before being delivered to the fusion module. A two-dimensional region of interest can be defined as the space $\mathcal{S} = [0, D_x] \times [0, D_y]$. With time discretization, the only difference between the new formulation and the original formulation is that Equation 3 becomes

$$G = \frac{\sum_{t=0}^T \sum_{j=1}^n u_j \bigvee_{i=1}^m \gamma_{ij}(t)}{T \sum_{j=1}^n u_j}, \quad (4)$$

where $t \in \{0, 1, 2, \dots\}$.

For the class of model-based algorithms, we use a discrete-time, discrete-state-space, stochastic model to express the target behavior and dynamics (as will be discussed in detail

¹Figure 3 is repeated in this chapter as Figure 6 for the reader's convenience.



(a) Type I: Low-rise, widely spaced UTZ.



(b) Type II: High-rise, closely spaced UTZ.

Figure 6: Two example UTZs (repeated).

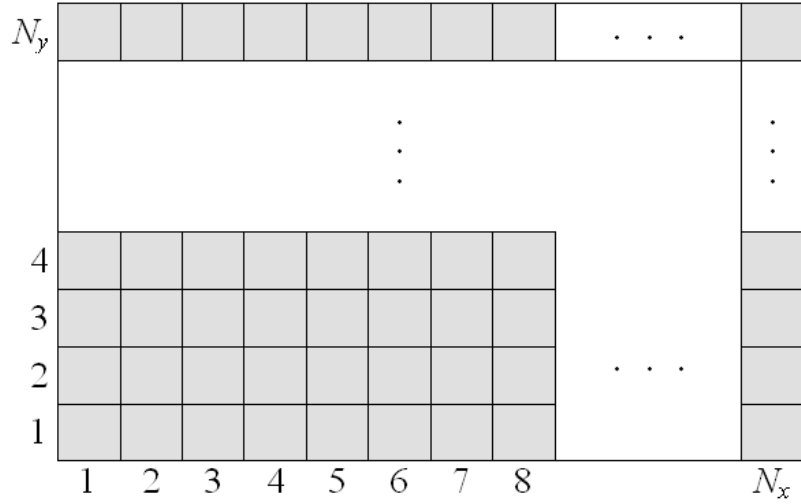


Figure 7: Grid-type region of interest.

in the following chapter). Consequently, the spatial region of interest \mathcal{S} is also discretized into cells. A two-dimensional $N_x \times N_y$ grid can be represented as

$$\mathcal{S} = \{x|x = 1, 2, \dots, N_x\} \times \{y|y = 1, 2, \dots, N_y\}. \quad (5)$$

Thus, a target position is expressed in terms of the cell in which it exists. A cell is represented by the pair (α, β) , where α takes values in $\{x|x = 1, 2, \dots, N_x\}$, and β takes values in $\{y|y = 1, 2, \dots, N_y\}$. Figure 7 shows an example of a grid-type discretized region of interest.

4.2.2 Reformulation for Type-II Urban Zones

In the previous subsection, we showed that an agent is free to move anywhere within \mathcal{S} for Type-I urban terrain zones. Extending the same approach to the three-dimensional space is possible thinking of buildings as obstacles that agents have to avoid. Notwithstanding, placing agents in the general three-dimensional space for monitoring a Type-II zone such as the one shown in Figure 6(b) may make it very hard to find general systematic solutions to the problem in real time, mainly because of the irregular nature of such environments. Buildings in Type-I regions are assumed to be of no effect on the navigation or the observation of an agent, whereas such assumption is not always valid in the case of Type-II regions. Therefore, there is a need for a general method of choosing the “best” observation points

for agents. In this thesis, the choice is made to limit the observation points to street intersections. The rationale behind this choice is that placing an agent at an intersection enables it to monitor all incident street segments as well as the vertical surfaces of the buildings on the sides of each street. Figure 8 shows a possible view from a mini-UAV hovering over a street intersection. As evident from the picture, the mini-UAV can easily observe all the moving vehicles on the street as well as the building surfaces all the way to the next street intersection. Street segments around an agent placed at a street intersection can be visually observed simultaneously using either multiple cameras or a single panoramic camera.



Figure 8: A possible view from a mini-UAV hovering over a street intersection.

Based on the previous discussion, the region of interest is better modeled as a graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \quad (6)$$

where \mathcal{V} is a set of graph vertices, such that each vertex v_i represents an observation point, and \mathcal{E} is a set of graph edges (or arcs), such that each edge e_i represents a street segment. Starting with a street map, \mathcal{G} can be created by placing vertices (observation points) at the street intersections. Extra observation points can be inserted as needed. For example, when a street segment is too long to be covered by a single agent's sensing range, one or

more observation points can be inserted along the segment to split that segment into two or more segments (see Figure 9).

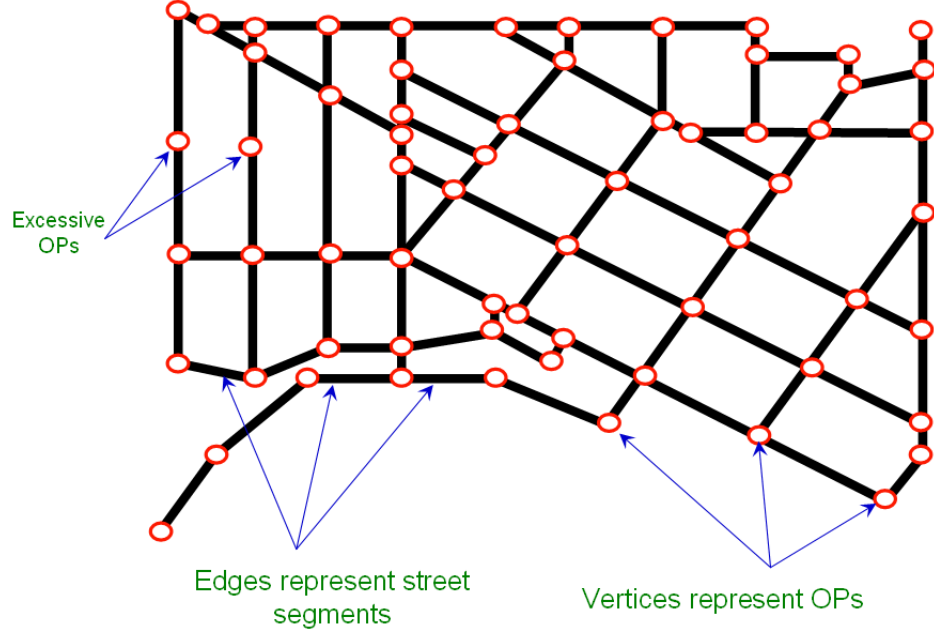


Figure 9: A graph representing Type-II urban terrain zone.

The agent placement problem in HC zones can thereupon be reformulated as follows:

Given:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: a graph representing a bounded, enclosed spatial region (as described above),
- $R = \{r_i | i = 1, 2, \dots, m\}$: a team of m agents that
 - are located on $\mathcal{U} \subseteq \mathcal{V}$, and
 - have observation sensors that are noisy and of limited sensing range (SR) that can cover the longest street segment represented in \mathcal{G} ,
- $O(t) = \{o_j(t) | j = 1, 2, \dots, n\}$: a set of n targets, such that, at time t , target $o_j(t)$ is located on some edge, $e_k \in \mathcal{E}$, and
- \mathbf{U} : a vector of utility values such that u_j reflects the utility value gained by observing target o_j .

Define an $m \times n$ Boolean matrix $\mathbf{\Gamma}(t)$, where

$$\gamma_{ij}(t) = \begin{cases} 1 & \text{if } o_j \text{ is detectable by } r_i \\ 0 & \text{otherwise} \end{cases}, \quad (7)$$

where “detectable” here means that the o_j is located on an edge that is incident to the vertex where r_i is located.

The goal is to provide a dynamic agent placement policy to maximize the normalized global utility function defined as

$$G = \frac{\sum_{t=0}^T \sum_{j=1}^n u_j \prod_{i=1}^m \gamma_{ij}(t)}{T \sum_{j=1}^n u_j}. \quad (8)$$

The reformulation discussed above only suites ground agents and aerial agents that possess the capability of hovering (e.g., a rotary-wing agent or a helicopter), in which case, all street segments incident to an intersection can be observed simultaneously. On the other hand, the reformulation may not be used for fixed-wing agents because a fixed-wing agent cannot observe multiple street segments at once since it cannot place itself over a street intersection for long enough periods of time. To reformulate the problem for the fixed-wing type, we can restrict an agent to fly over a single street segment for each sampling period instead of hovering over an intersection. In this case, the agent speed will vary accordingly with the length of the segment it is flying over. Under the assumption that the agent can only observe a single street segment at a time, the problem becomes a much simpler one. It only entails searching for the most important segments and flying agents over those segments. Therefore, an optimal solution can be found by flying the m fixed-wing agents over the m street segments that contain the most important targets. In other words, a simple greedy policy, which sorts the street segments according to the sum of the utility values of the targets they contain, constitutes an optimal policy in this case. In scenarios where both types of agents need to be considered, the problem can be solved in two steps. First, the problem can be solved for the rotary-wing (hovering-capable) agents. Then, the fixed-wing agents can be flown over the most important street segments that have not been covered by the rotary-wing agents. Again, the difficult component of the problem is the one

that deals with the hovering-capable agents. Therefore, this thesis is primarily interested in studying the harder case of the agent placement problem in Type-II urban terrain zones using rotary-wing agents that possess the capability of hovering. This is discussed in detail in Chapter 6.

CHAPTER V

AGENT PLACEMENT FOR TYPE-I URBAN ZONES

This chapter shows how to apply the distributed approach presented in Chapter 4 to Type-I urban terrain zones. As the approach suggests, two classes of algorithms (non-model-based and model-based) are to be used. An example algorithm of each class is developed. Then, a novel approach to online target motion model learning is presented.

5.1 Non-Model-Based Algorithms

5.1.1 Main Assumptions

While developing the proposed algorithm, the following main assumptions have been made:

- **A1** Each agent is equipped with processing and communication infrastructure,
- **A2** Each agent is equipped with real-time target recognition software that enables real-time identification of the moving targets,
- **A3** The processing and inter-agent communication delays are negligibly small compared to the targets' as well as the agents' dynamics, and
- **A4** The agents possess low-level control modules with collision avoidance capabilities.

The assumptions stated above are believed to be reasonable given the technology available at the current time. **A1** is satisfied in many real-life agents, such as robots, UAVs, and UGVs. **A2** is concerned with target tracking applications that usually require not only target detection, but also target identification and classification. As mentioned in Chapter 4, recent advances in many research areas including computer vision (e.g., [13] and [26]) have made it possible to achieve those tasks in a timely manner. **A3** implies that no significant changes in the positions of targets or agents will occur while the agents' next best locations are being computed. **A4** limits the job of the algorithm to determining the next best location of an agent. Generating a path to move the agent to that location is carried

out through low-level control algorithms that include collision avoidance capabilities. The low-level motion control is beyond the scope of this doctoral thesis.

5.1.2 Algorithm Description

The proposed algorithm requires each agent to be aware of other agents that may exist in its vicinity, possibly by broadcasting periodic “hello!” messages (known as *heartbeating* in computer networking jargon). The current x - y positions of an agent and any other information to be broadcast can be piggybacked on the periodic heartbeating messages. Agents periodically share their local target information with their neighboring agents whenever possible. In some cases, not all targets detected by an agent r_i may be of interest to a neighbor agent r_k since some of those targets maybe too far from r_k . Thus, r_k chooses to ignore the information about the targets that are more than $S_k + D_k$ units away, where S_k is the sensing range of r_k and D_k is the maximum distance that can be traveled by r_k in a sampling period. This excludes the targets that the agent cannot detect before the beginning of the next sampling period. For each agent r_i , we call the set of targets falling within a distance of $S_i + D_i$ from r_i , the set of *reachable targets*, and we call the set of reachable targets being observed locally or remotely (through other agents), the set of *observed reachable targets*, $\Omega_i(t) \subseteq O(t)$. Both sets are time-dependent; we include the time-dependence notation only when necessary. Given the set of observed reachable targets, each agent computes the center of mass of the targets using their utility values as the masses

$$(x_{c,i}, y_{c,i}) = \frac{1}{\sum_{o_j \in \Omega_i} u_j} \left(\sum_{o_j \in \Omega_i} u_j x_j, \sum_{o_j \in \Omega_i} u_j y_j \right), \quad (9)$$

where $(x_{c,i}, y_{c,i})$ is the location of the center of mass computed by r_i , (x_j, y_j) is the location of target o_j , and $\Omega_i \subseteq O$ is the set of all observed reachable targets associated with r_i . Each agent r_i will then move to the computed location; if the location is more than D_i units away from the current agent location, the agent will move as close as possible to the computed location. The rationale behind the center-of-mass approach is that the higher the utility value of a target is, the closer the center of mass will be to that target, and hence, the less of a chance the targets with high utility values will escape the surveillance of the agent.

It is possible that two or more agents keep following the same target, especially a target with a high utility value. This is likely to happen since the center of mass approach urges agents to observe high-utility targets and possibly ignore low-utility ones. In this case, each agent will have optimized its own local utility (the sum of utility values of targets under local surveillance) at the expense of the global utility. To avoid such situations, agents have to coordinate their decisions in order to avoid tracking already tracked targets. It is necessary for each agent to have a unique ID (e.g., the physical address of its communication interface). Each agent broadcasts the set of targets that will be observed when the agent moves to the new location. If an agent learns that a target will be monitored by another agent with a lower ID, it excludes that target from the center-of-mass computation. This guarantees that no target will be tracked by two or more agents at the expense of the global utility. If an agent has already included a target that will be monitored by another agent with a lower ID, it can be cancelled out as follows:

$$(\hat{x}_{c,i}, \hat{y}_{c,i}) = \frac{1}{u_{c,i} - u_e} (u_{c,i}x_c - u_e x_e, u_{c,i}y_c - u_e y_e), \quad (10)$$

where $(\hat{x}_{c,i}, \hat{y}_{c,i})$ is the refined center of mass, the subscript $_e$ refers to the target to be excluded, and $u_{c,i}$ is the sum of the utility values of the observed reachable set of targets. Equation 10 is equivalent to inserting an imaginary point mass with a negative value at the same location of the target to be excluded. This saves the agent from re-computing the new center of mass from scratch. An agent can iteratively repeat Equation 10 for each target to be excluded.

Finally, it may be desired to put the *next desired agent location* in a closed form. For that purpose, we define the following two time-dependent sets:

- *locally observed targets*, $\Psi_i(t) = \{o_j(t) \mid \gamma_{ij}(t) = 1\}$: the set of targets falling within S_i at time instant t ($\gamma_{ij}(t)$ is defined in Equation 2), and
- *exclusive observed reachable targets*, $\Phi_i(t)$: defined as follows:

$$\Phi_i(t) = \Omega(t) \setminus \bigcup_{k < i} \hat{\Psi}_k(t+1), \quad (11)$$

where $\hat{\Psi}_k(t+1)$ is the predicted set of locally observed targets for r_k . Also, in this equation, the subscripts i and k represent the unique IDs discussed above.

Thus the closed form for the next desired location for an agent r_i is

$$(x_{c,i}, y_{c,i}) = \frac{1}{\sum_{o_j \in \Phi_i} u_j} \left(\sum_{o_j \in \Phi_i} u_j x_j, \sum_{o_j \in \Phi_i} u_j y_j \right), \quad (12)$$

where Φ_i is defined in Equation 11.

The algorithm is named the Center-of-mass-based Exclusive Placement (CEP) algorithm since it places an agent at the center of mass of targets that it can observe exclusively. Figure 10 shows a pseudo-code description for CEP that runs on each agent r_i . CEP is a distributed algorithm and hence can easily tolerate agent failures. At those steps where the algorithm has to wait on messages from other agents, CEP uses timeouts to avoid waiting indefinitely for messages from a failed agent. This is illustrated using the clock symbol “🕒” next to the steps that are timed. If the timeout clock expires before receiving an expected message from a certain agent that has possibly failed, r_f , the algorithm will conclude that r_f has failed at this point. Nevertheless, the algorithm can still function in the absence of the failed agents.

5.1.3 Algorithm Complexity

Referring to Figure 10, the worst-case complexity of each step can be analyzed as follows:

- **Step 1:** $O(n)$; worst case corresponds to all n targets having been detected and broadcast.
- **Step 2:** $O((m-1)n) = O(mn)$; worst case is determined assuming that all the other $m-1$ agents are able to communicate with the current agent, and that each of those agents has detected and broadcast all the n targets.
- **Step 3:** $O(n)$; a single scan over the list of targets (n targets in the worst case) can determine which targets fall within $S_i + D_i$ distance from the agent.
- **Step 4:** $O(n)$; in the worst case, an agent will include all the n targets in computing the center of mass, in which case, Equation 9 incurs

1. Broadcast the set of all observed targets Ψ_i and their locations.
- 🕒 2. Wait on a message containing Ψ_k from each neighbor r_k .
3. Determine the set of observed reachable targets, Ω_i .
4. Compute center of mass: $(x_{c,i}, y_{c,i}) = \frac{1}{\sum_{o_j \in \Omega_i} u_j} \left(\sum_{o_j \in \Omega_i} u_j x_j, \sum_{o_j \in \Omega_i} u_j y_j \right)$.
- 🕒 5. Wait on a message from each neighbor r_k with $k < i$; message will contain a set of targets, $\hat{\Psi}_k$, that will be covered by r_k .
6. $\Psi = \bigcup_{k < i} \hat{\Psi}_k$.
7. For each $o_e \in \Psi$, refine the center of mass position by excluding o_e :
 $(x_{c,i}, y_{c,i}) = \frac{1}{u_{c,i} - u_e} (u_{c,i} x_c - u_e x_e, u_{c,i} y_c - u_e y_e)$.
8. Determine and broadcast the set $\hat{\Psi}_i$ of targets that will be covered by moving to $(x_{c,i}, y_{c,i})$.

Figure 10: CEP pseudo-code representation.

- $2n$ multiplication operations (n operation for the x component and n operations for the y component),
- $3n - 3$ addition operations ($n - 1$ operation for each of the following: the numerator of the x component, the numerator of the y component, and the denominator which is common for the x and the y components), and
- two division operations.

Thus, the number of operations involved in computing the center of mass is $5n - 1$, which leads to $O(n)$ complexity.

- **Steps 5 and 6:** $O(mn)$; assuming each of the other $m - 1$ agents has sent a message with all the n targets, in which case, building Ψ involves scanning $m - 1$ lists, each of which contains n elements.
- **Step 7:** $O(n)$; that is assuming all the n targets will be excluded and using a similar argument to the one used in **Step 4**.

- **Step 8:** $O(n)$; assuming all the n targets will be observed from the new location.

Adding up the individual complexities of each of the steps, the overall worst-case complexity can be shown to reduce to $O(mn)$. This means that the algorithm scales well with the increase in the number of targets as well as the number of agents.

5.2 *Model-Based Algorithms*

The assumptions used for the model-based algorithms are the same as defined in the previous section, in addition to the fact that target motion follows some stochastic model. In this section, the target motion model is first defined. The justification of the model choice is discussed. Then, the model-based algorithm is presented and analyzed.

5.2.1 Target Motion Model

The target motion model to be used by the distributed approach described in Chapter 4 is required to

- **R1** capture both the random nature and the dynamics of target motion,
- **R2** make no assumptions regarding linearity since targets of interest are not generally expected to adhere to a linear model,
- **R3** be easily learned from fused sensor observations since the approach suggests that a model building task will take place,
- **R4** have a parsimonious representation, so it can be learned online, and
- **R5** support distributed learning and fusion.

In the rest of this section, we present the model of choice and discuss how it can address the requirements listed above (except for requirements **R3** and **R5**, which will be discussed in Section 5.3).

A stochastic model is used to describe the motion of each type of targets. The region of interest can be divided into cells as shown in Figure 7. For each type of targets, the cells in which a target cannot exist (e.g., a vehicle cannot be driven over building roofs) are

excluded. The cell size is determined based on many factors including the agents' sensing range, the maximum target and agent speeds, etc.

To better illustrate the model, consider the following example. Assume that, at time $t - 1$, a target o_k was in a cell (α, β) . Further, assume that, between time $t - 1$ and t , o_k had the full freedom to choose between remaining in the same cell and moving to one of the eight neighbor cells with different probabilities. Formally, let $\chi_k(t - 1)$, $\chi_k(t)$ express the location of o_k at time $t - 1$ and time t , respectively. Thus, $\chi_k(t)$ depends only on $\chi_k(t - 1)$ since the target has full freedom to choose which location out of the nine locations to move to. Because we do not know what location the target will choose, we treat $\chi_k(t)$ as discrete-space random variable (since it takes values in a discrete set, $\mathcal{S} = \{x|x = 1, 2, \dots, N_x\} \times \{y|y = 1, 2, \dots, N_y\}$), meaning that χ_k is a discrete-space discrete-time random process. In view of the fact that, at time t , $\chi_k(t)$ depends only on $\chi_k(t - 1)$, we conclude that the location of a target of that kind obeys a first-order Markov chain [27] such that

$$P(\chi_k(t) = c_i) = P(\chi_k(t) = c_i | \chi_k(t - 1) = c_j) = q_{ij}, \quad (13)$$

where c_i and c_j are any two cells, and q_{ij} is the one-step transition probability from cell c_j to cell c_i . For the specific example given above, q_{ij} is strictly zero only if c_i and c_j are not *eight*-neighbors.

Let $\mathbf{X}_k(t - 1)$ be a vector representing the probability distribution (over \mathcal{S}) of the target o_k at time $t - 1$. $\mathbf{X}_k(t - 1)$ has as many components as the cardinality of \mathcal{S} . Each component represents the probability of o_k being in the corresponding cell. Using this model, the cell probability distribution of the target location at time t , can be estimated ahead of time at time $t - 1$ as

$$\hat{\mathbf{X}}_k(t) = \mathbf{Q}\mathbf{X}_k(t - 1), \quad (14)$$

where \mathbf{Q} is the transition probability matrix¹ such that q_{ij} is defined above. As an example from real life, assume some vehicle traveling north is currently (at time t) in cell c_i corresponding to a road intersection. There are different scenarios that can take place before

¹Note that this thesis uses Bartlett's convention, in accordance with which, the columns (not the rows) of a transition matrix sum up to 1.

$t + 1$; the vehicle may (1) stay in the same cell, e.g., if the light is red, (2) go straight (go north), (3) turn right (go east), (4) turn left (go west), or (5) turn around (go south). Thus the only nonzero entries in the i^{th} column of \mathbf{Q} are those corresponding to c_i and to the cells that are located north, east, south, and west of c_i . Note that, in general, each cell in \mathcal{S} has different transition probabilities from other cells.

Now, let us consider a more interesting version of the example described above. Assume the same vehicle was moving north at, say, 40 miles per hour as it reached the cell where the intersection is. The dynamics of the vehicle would suggest that the most-likely scenario to occur before the next time step is that the vehicle will be at the cell north of the intersection's cell. This can be expressed as a high probability at the corresponding entry in the transition matrix. Further, it is very unlikely that the vehicle will make a turn at that relatively high speed. This unlikeliness can be expressed by very small probabilities at the relevant locations in the transition matrix. Furthermore, it is extremely unlikely or impossible that the vehicle can turn around all of a sudden, which can be expressed by a zero transition probability to the cell south of the intersection's cell. This means that the high-speed case violates the model given above since the probability distribution of the vehicle location depends not only on the vehicle's current location, but on the speed as well. Given that speed is deducible from the last two locations of the vehicle, this means that the probability distribution of the future vehicle location at a certain time depends on the last two locations. To complicate the situation further, we may argue that the probability can also depend on how much the vehicle was accelerating or decelerating as it entered the cell. Using a similar argument, the distribution under consideration depends on the last three vehicle locations. The same argument can go on for higher-order motion derivatives. The higher the order of the derivative to be considered is, the more steps we have to look back when computing the distribution.

Consequently, from the discussion provided above, we conclude that a first-order Markov chain can model the random component of a target motion, but it seems to fall short when it comes to modeling the dynamics of a target, especially one with heavy dynamics. To allow the model to accommodate both the random behavior and the dynamics associated with

target motion, we consider a high-order Markov chain [36] [11], where M^{th} -order Markov chain obeys

$$P(\chi_k(t) = c_i) = P(\chi_k(t) = c_i | \chi_k(t-1) = c_{j_1}, \dots, \chi_k(t-M) = c_{j_M}). \quad (15)$$

Thus, a model of the variety of M^{th} -order Markov chain can address requirement **R1** stated above. As for requirement **R2**, it has been shown [4] [3] that a high-order Markov process can generally model a nonlinear difference equation on the form

$$\xi(t+1) = f\{\xi(t), v(t), \epsilon(t)\}, \quad (16)$$

where $f\{\cdot\}$ is a nonlinear function, $\xi(t)$ is the state vector, $v(t)$ is the input vector, and $\epsilon(t)$ is an independent Gaussian random vector. The order of the representing Markov chain depends on the order of Equation 16.

Presenting a high-order Markov chain parsimoniously has been studied by several researchers including Raftery who introduced the widely known *Mixture Transition Distribution* (MTD) in 1985 [36]. He also showed through three examples involving real data that MTD can model real data more successfully than other models. The MTD model avoids the exponential increase in the model independent parameters [36] [11] that occurs if we try to model the high-order Markov chain as a first-order Markov chain by augmenting \mathbf{X} and \mathbf{Q} defined in Equation 14. Therefore, MTD was introduced as a parsimonious model that adds only one independent parameter for every increase in the model order above the first order. Mathematically, the model can be expressed as (note that the subscript k representing the target index is dropped for simplicity)

$$\begin{aligned} \mathbf{X}(t) &= \sum_{i=1}^M \lambda_i \mathbf{Q} \mathbf{X}(t-i); \\ \sum_{i=1}^M \lambda_i &= 1, \end{aligned} \quad (17)$$

where λ_i describes the contribution of each lag. Thus, λ_i is the only independent parameter added to the model as the model order is increased.

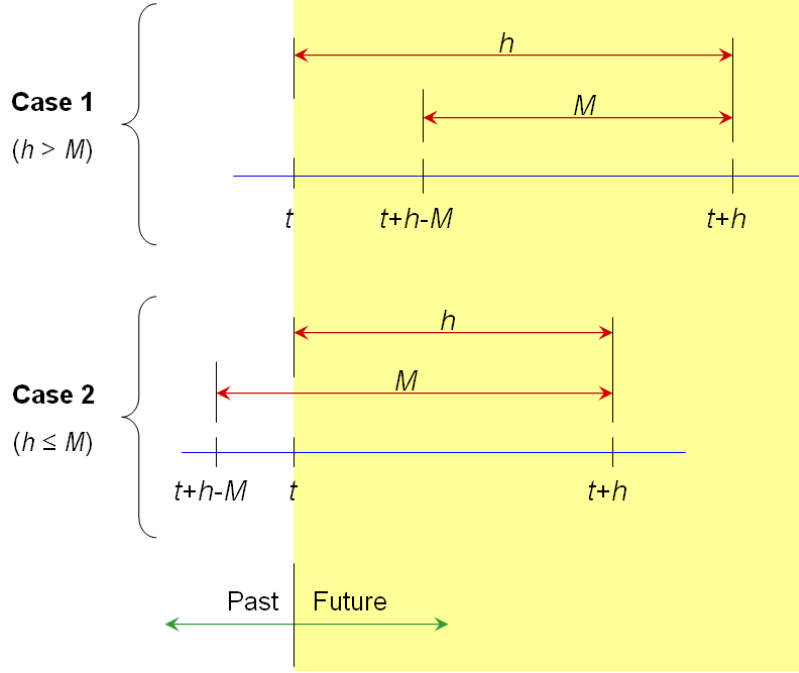


Figure 11: Predicting target location using a high-order Markov chain.

A more general MTD model, called *Generalized Mixture Transition Distribution* (GMTD) [11], introduces a whole transition matrix \mathbf{Q}_i for each lag such that

$$\begin{aligned} \mathbf{X}(t) &= \sum_{i=1}^M \lambda_i \mathbf{Q}_i \mathbf{X}(t-i); \\ \sum_{i=1}^M \lambda_i &= 1. \end{aligned} \quad (18)$$

Although the GMTD model bears more parameters than the MTD model, both models are considered parsimonious since they avoid the exponential blow up in the number of model parameters that would occur with a traditional model. Thus both models can address requirement **R4** listed above. We suggest that both models can represent the target motion in the problem under consideration.

5.2.2 Model-Based Algorithm Description

Based on the GMTD model described in the previous subsection, at time t , an agent r_k can compute the probability distribution of the location of a target o_j at a future time instant

$t + h$ as

$$\hat{\mathbf{X}}_j(t+h) = \begin{cases} \sum_{i=1}^M \lambda_i \mathbf{Q}_i \hat{\mathbf{X}}_j(t+h-i) & \text{if } h > M \\ \sum_{i=1}^{h-1} \lambda_i \mathbf{Q}_i \hat{\mathbf{X}}_j(t+h-i) + \sum_{i=h}^M \lambda_i \mathbf{Q}_i \mathbf{\Xi}_j(t+h-i) & \text{if } h \leq M \end{cases}, \quad (19)$$

where $\mathbf{\Xi}$ is a vector describing the best estimate of the location at past time instants where fused sensor data are already available. Equation 19 handles two different cases: $h > M$ and $h \leq M$ (see Figure 11). The first case happens if the prediction is carried out for a time instant that is very far ahead in the future compared to the model order, in which case, the prediction depends only on other predictions since all of the lags will still be in the future. The second case, on the other hand, corresponds to situations where the prediction is carried out for a relatively near time instant compared to the model order. In this case, an agent should have already obtained fused sensor observations for some of the lags, thus the use of $\mathbf{\Xi}$. Note that in most cases, $\mathbf{\Xi}$ is expected to represent an impulse probability distribution over \mathcal{S} . In other words, it is known exactly which cell the target was in at that time instant. This becomes the case especially if the underlying sensor uncertainty bounds are smaller than the cell size, into which \mathcal{S} is discretized. However, no assumption is made regarding the shape of the distribution while the algorithm is designed, and the estimation of the target location based on sensor measurements (using Kalman filters or otherwise) is assumed to be carried out by the lower-level sensor fusion module, which, according to Chapter 4, is beyond the scope of this doctoral thesis.

Then, we define a quantity called *probable cell outcome* for each cell c over a future time horizon H in terms of both the probability of targets existing in c during H and the utility values of targets u_j as

$$\eta_c(H) = \sum_{o_j \in \mathcal{O}} u_j \left(\sum_{h=1}^H v_h \left(\hat{\mathbf{X}}_j(t+h) \cdot \mathbf{Y}_c \right) \right);$$

$$\sum_{h=1}^H v_h = 1, \quad (20)$$

where v_h describes the contribution of the future location prediction at time $t + h$ to the placement decision, and \mathbf{Y}_c is a binary vector with all zeros and a one at the location corresponding to the cell c . The equation sums up the *probability* \times *utility* quantities over

all observed targets and over a future horizon H .

Every F time units (where, in general, $F \leq H$), an agent r_i computes its next best location as the center of mass of the cells it can cover with probable cell outcomes used as masses. The rationale behind using the center-of-mass approach again is that the center of mass is known to be closer to cells with high probable outcomes, which are the cells that will most likely contain important targets with high probabilities. Placing an agent close to those cells minimizes the chance that those targets will escape surveillance. Note also that the center of mass can be computed in linear time with respect to the number of cells included in the computation, making it an attractive approach in terms of computational requirements. Figure 12 shows a typical map of probable cell outcomes. The outcomes are shown as “clouds” around targets, which are expressed as a ‘T’s. A brighter cell has more probable outcome than a darker one. Agents (expressed as ‘A’s) place themselves over the center of mass of the cells using their probable outcomes as masses. The circles in the figure indicate the sensing range of each agent.

The model-based algorithm, named Predictive Center-of-mass-based Exclusive Placement (PCEP) algorithm, can be defined from CEP (described in Section 5.1.2) by replacing “target” with “cell” and “target utility” with “probable cell outcome”. The attribute “predictive” is due to the ability of the algorithm to predict future target locations as probability distributions. The philosophy behind the center-of-mass approach and the intention of monitoring cells exclusively by a single agent remain the same as discussed for CEP.

5.3 Online Model Building

Figure 13 shows the proposed model building approach. A subset of the fused sensor observations is used to estimate the model parameters while the rest of the observations are used for model evaluation. The model estimation is done in two steps. The first step delivers a rough version of the model in order to meet the real-time requirements for the online model estimation. The estimation process goes on until the model achieves a minimum acceptable accuracy level. As soon as that level is reached, the model is released. At this point, agents can switch to using the model-based algorithm. In the meantime, the

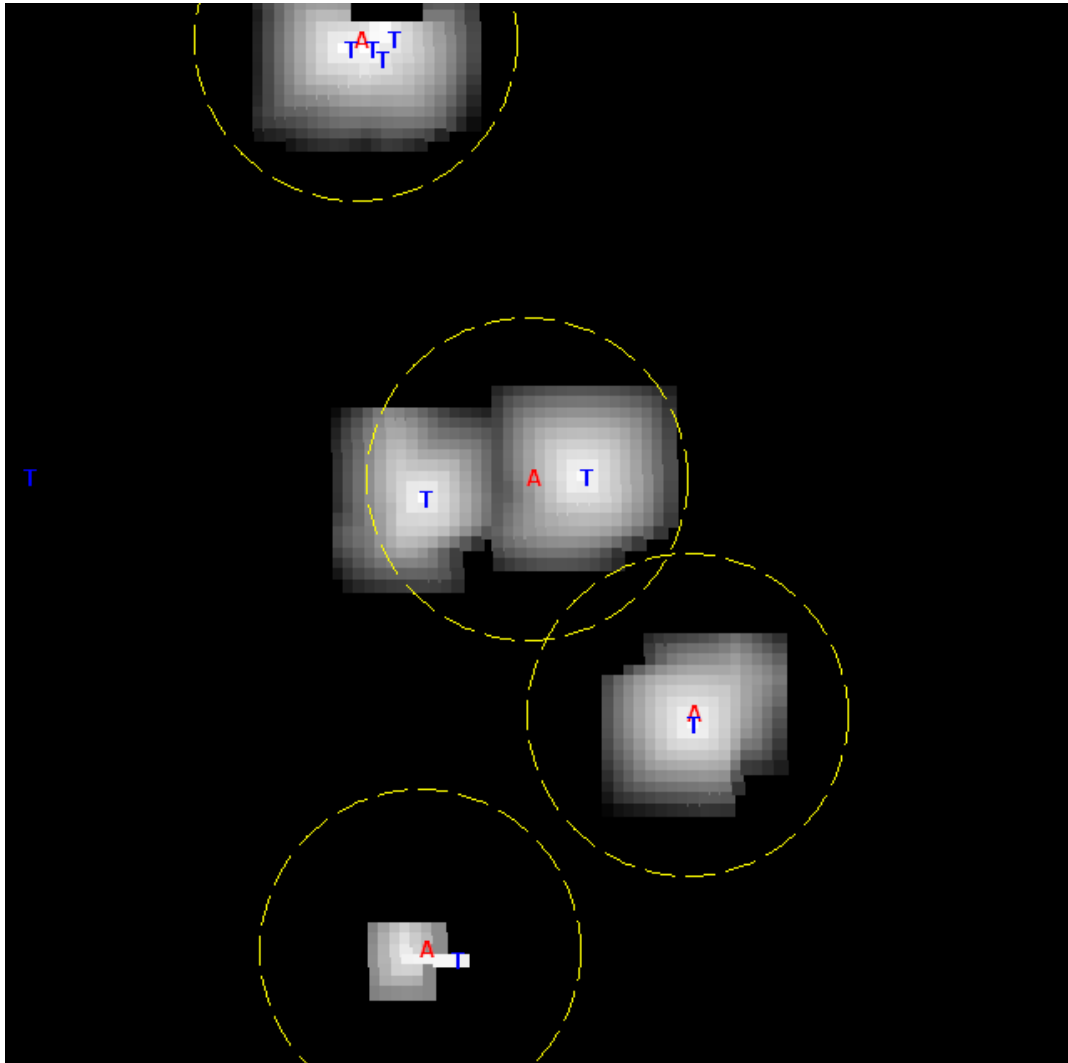


Figure 12: Model-based agent placement using the probable cell outcomes.

model can be further optimized to achieve higher accuracy levels (the second step). Model optimization is likely to be a time-consuming process depending on the model complexity and the desired accuracy level. As a result, it is set to run as a background process. Note that the optimization module is optional and the approach still works since the first step meets the minimum acceptable accuracy requirements.

5.3.1 Order Estimation

One of the most common methods for estimating the order of an MTD-type model is by optimizing some information-theoretic quantity such as Akaike information criterion (AIC)

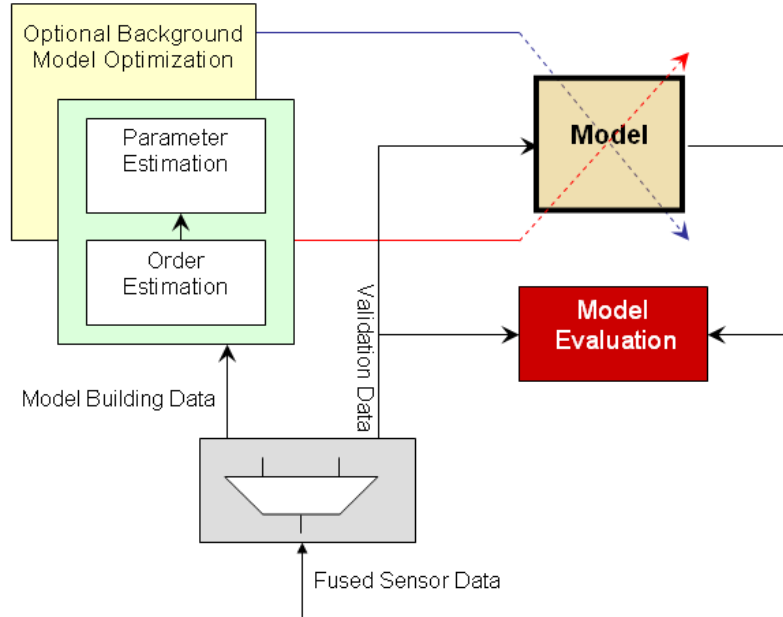


Figure 13: Overview of the model building approach.

or Bayesian information criterion (BIC) as shown in [11]. The main idea is to start off with the assumption that the model is a first-order model and use some optimization technique to come up with the optimal (or sub-optimal) model parameters. Based on the estimated parameters, the information criterion, AIC or BIC, is computed for the first-order model. Then, the process is repeated for the assumption of a second-order model. Then, AIC or BIC is computed again, and so on. Every time the model order is increased, the AIC or BIC is expected to improve until a diminishing-return point is reached, based on which the model order can be decided.

The main disadvantage of the AIC/BIC-optimization method described above is that it is extremely slow and hence unsuitable for real-time applications such as the one under consideration. For example, to reach a conclusion that some collected observations belong to a fifth-order Markov chain, it would require to estimate/optimize six different models. Building a model online for a real-time application requires a fast method that can possibly estimate the order in “one shot”. Therefore, This thesis proposes using the partial autocorrelation function (PACF) [44] [1] for model order estimation, which is a widely used method in time-series analysis. The PACF of a sequence Z evaluated at lag l can be defined as

the autocorrelation between $z(t)$ and $z(t - \ell)$ after removing the effect of lags 1 through $\ell - 1$. An example plot of a PACF is shown in Figure 14. The model order can be estimated according to a desired confidence level.

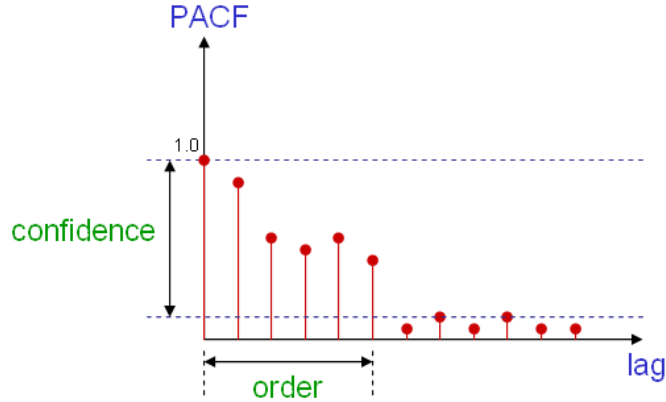


Figure 14: Estimating model order from partial autocorrelation plot.

The basic autocorrelation function is generally computed for a one-dimensional sequence of observations. In our case, each cell is represented by two different components: an x component and a y component. Consequently, the sequence of sensor observations results in two sequences, which will be treated separately. The order of the model is estimated as the maximum of the two orders obtained by processing each of the two sequences separately.

5.3.2 Parameter Estimation

Berchtold has developed an iterative optimization algorithm for estimating the MTD model parameters. Berchtold’s algorithm performs at least as good as or better than other existing algorithms [10]. This is the approach we adopt for model estimation. The main drawback of this algorithm, as far as the application is concerned, is the fact that it may take a considerably long time to converge, especially since the estimation technique in our case may have to deal with very large transition matrices. Thus, a modified version of Berchtold’s algorithm is proposed where the severity of this drawback is considerably reduced. To better understand the modification, consider the example illustrated in Figure 15. The left-hand side of the figure shows a part of a certain 50×50 -cell region of interest with all possible

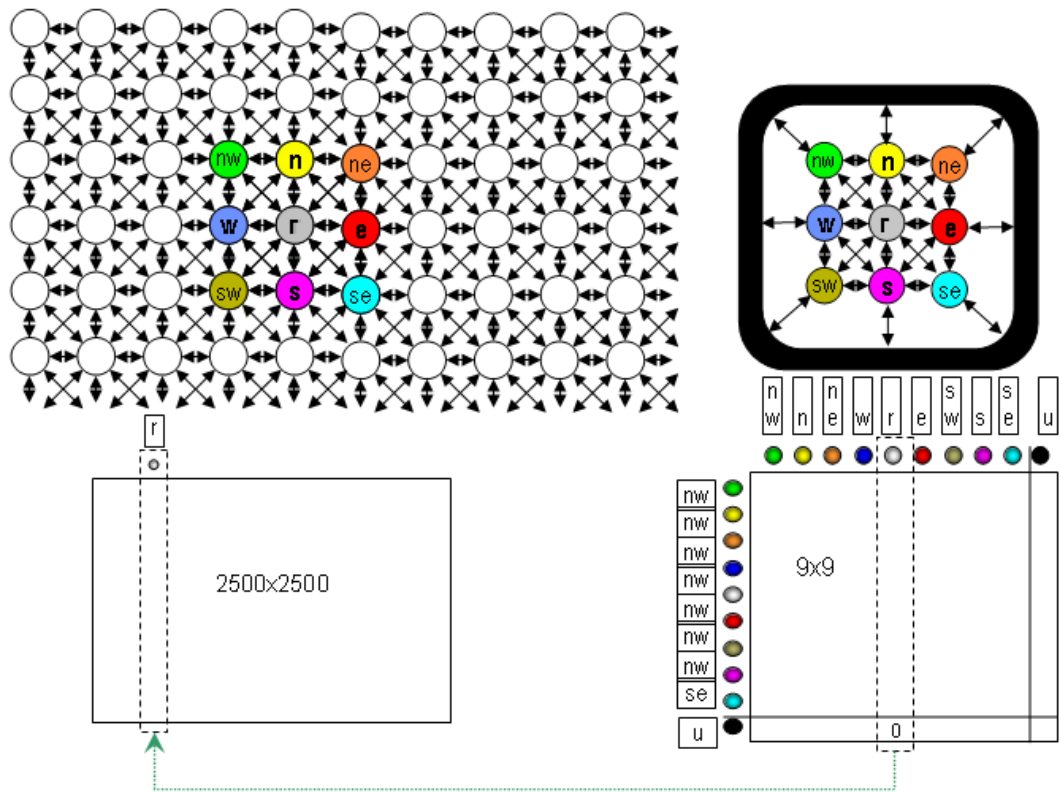


Figure 15: Modified Berchtold's algorithm for estimating a transition matrix.

transitions that a certain target can make. Each cell is represented by a circle, and the inter-cell transitions are represented by arrows (transitions corresponding to the target staying in the same cell are not shown for clarity). Since the target has a limited speed, it can only move to those cells that are in a small neighborhood of its current cell. Suppose, in this particular example, that a target can either stay in the same cell or move to one of the eight neighboring cells in a single transition. If we assume that a target being in cell c_r corresponds to a target state c_r , then cells other than the current cell and its eight neighbors correspond to a single unreachable state c_u , i.e., a state that cannot be reached by a single transition given the current target location. Berchtold’s algorithms can then be applied to optimize the parameters associated with the new setup (shown on the right-hand side of the figure). In this case the algorithm needs to optimize 2,500 10×10 matrices instead of optimizing a single large $2,500 \times 2,500$ matrix. The column that corresponds to the current cell c_r will contain the actual transition probabilities, and those probabilities will be placed in the relevant locations in the full matrix. The algorithm is repeated for all of the cells the same way it is applied to c_r . The modified Berchtold’s algorithm hence addresses requirement **R4** (see Section 5.2.1).

Not only does this modified approach reduce the computational complexity of the algorithm for the problem of interest, it also enables the algorithm to be parallelized at two different levels. The first level of parallelization can be done in a SPMD² [16] fashion, by having each agent compute the transition probabilities for the part(s) of the region of interest that it has explored (requirement **R5** in Section 5.2.1). The second level of parallelization is concerned with the computational load within a single agent, assuming that its computing infrastructure is a multi-processor system supporting the SIMD³ model [16], in which case, each processor can be charged with estimating a subset of the transition matrix columns that the agent, as a whole, is required to estimate.

²Single-program-multiple-data-streams.

³Single-instruction-stream-multiple-data-streams.

CHAPTER VI

AGENT PLACEMENT FOR TYPE-II URBAN ZONES

This chapter discusses in detail how to apply the distributed approach presented in Chapter 4 to the high-rise, closely spaced urban terrain zones. Recall that Section 4.2.2 argued that searching the entire three-dimensional space of a Type-II urban terrain zone is likely to make it rather difficult to develop a general systematic solution to the agent placement problem. The decision is made to limit the search space to the street intersections. An agent placed at a street intersection can observe all incident street segments as well as building surfaces on both sides of each segment. Thus, Section 4.2.2 reformulated the problem such that the region of interest is modeled as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} is a set of graph vertices such that each vertex v_i represents an observation point (OP), and \mathcal{E} is a set of graph edges (or arcs) such that each edge e_i represents a street segment (refer to Section 4.2.2 for more details). Section 4.2.2 also argued that this reformulation, which deals with hovering-capable agents, constitutes a harder problem than the one of dealing with fixed-wing agents that cannot hover.

As proposed in Chapter 4, the distributed approach entails two classes of algorithms: non-model-based and model-based. The following sections introduce an instance of each class.

6.1 Non-Model-Based Algorithms

6.1.1 Assumptions and Background

It is assumed that agents are provided with a street map of the region of interest as part of the information received from the situation awareness module described in Section 1.3.1. Agents can then build the relevant data structures that encode the graph \mathcal{G} , which represents the region of interest. Every sampling period t , each agent r_i broadcasts its location $v_i(t) \in \mathcal{V}$, the set of locally observed targets $\Psi_i(t)$ (targets that are located on the street segments

incident to the street intersection where the agent is currently located), and the edges where the targets are located. Each agent will then build a weighted version of the graph

$$\mathcal{H} = (\mathcal{V}, \mathcal{E}, \boldsymbol{w}), \quad (21)$$

where $\boldsymbol{w} : \mathcal{E} \rightarrow \mathbb{R}$. Note that since targets change their locations over time, the edge weights \boldsymbol{w} are time-dependent, and so is \mathcal{H} . However, the time-dependence notation is left out for clarity; it is understood that agents update the weights every discrete-time instant. The weight of an edge e is given by

$$w(e) = \sum_{o_j \text{ on } e} u_j, \quad (22)$$

i.e., the sum of the utility values of all of the targets located on an edge.

The algorithm attempts to maximize the coverage defined in Equation 8 by searching for a set of OPs, $\mathcal{U} \subseteq \mathcal{V}$, that maximizes the coverage at each time step, which can be defined as

$$g = \sum_{e \in \mathcal{E}} \text{adj}_{\mathcal{U}}(e) w(e), \quad (23)$$

where $\text{adj}_{\mathcal{U}} : \mathcal{E} \rightarrow \{0, 1\}$, such that $\text{adj}_{\mathcal{U}}(e)$ is 1 if and only if e has at least one end in \mathcal{U} . This is known as the *max vertex cover problem*, which is an NP-hard problem [19]. Consequently, we are only interested in approximate solutions.

Recent literature exhibits a number of approximate solutions to the max vertex cover problem with guaranteed lower performance bounds. One of the best-known polynomial-time solutions is based on a linear programming (LP) relaxation of the problem [2] with an approximation ratio of $\frac{3}{4}$. A simple greedy algorithm [19] has a performance guarantee of $\frac{m}{|\mathcal{V}|}$ of the optimal solution, which would be very low for the cases considered in our application since the number of agents is expected to be much less than the number of street intersections, not to mention the excessive OPs that are inserted on demand. An iterative greedy algorithm [20], provides a performance guarantee of $1 - e^{-1}$, which is about 84% of the performance guarantee of the solution provided by the LP solution¹.

The LP approximation is an attractive option since it provides a good approximation in polynomial time, and thus is expected to deliver fast solutions using a centralized LP solver.

¹ e here refers to the natural logarithm base (≈ 2.7183).

Nonetheless, since this doctoral thesis is interested in developing distributed solutions to the agent placement problem, the only way to adopt the LP approach is via decentralizing the LP solver. LP decentralization has two basic disadvantages from a distributed system’s point of view. First, LP decentralization may use up the communication bandwidth since agents have to exchange a large number of messages before obtaining a solution of comparable quality to the centralized LP solution. Also, note that applying an LP relaxation to an integer programming problem such as the max vertex cover problem entails applying a rounding algorithm used (after solving the linear program) to obtain the integer solution. This adds to the number of messages that need to be exchanged among agents. Second, a decentralized LP solution is generally only an approximation to the centralized LP solution. For example, Bartal *et al.* [5] studied a distributed LP solution to a minimization problem. They showed that their problem can be solved in a distributed fashion within a $(1 + \varepsilon)$ approximation ratio² of the centralized LP solution, with a number of communication rounds that increases with at least $\frac{1}{\varepsilon^4}$. This closes the gap even more between the LP relaxation and the iterative greedy approximation discussed above, making the iterative greedy algorithm more attractive overall.

6.1.2 Algorithm Description

An iteration of the iterative greedy algorithm starts by assigning weights to the vertices such that the weight of a vertex v is the sum of the weights of all incident edges, i.e., $w(v) = \sum_{adj_{\{v\}}(e)=1} w(e)$. Then, it picks the vertex v_{max} with the maximum weight and adds it to the cover set. The vertex v_{max} and the incident edges are then removed from the graph. The whole process is repeated in the next iterations until m vertices are obtained. Thus, the algorithm finds a solution in m iterations. The algorithm can be decentralized by dividing the set of vertices \mathcal{V} into m disjoint subsets; each agent r_i is assigned a subset \mathcal{V}_i (see Figure 16). Note that, in general, there are edges connecting some vertices from \mathcal{V}_i with some vertices from \mathcal{V}_j for any $j \neq i$.

The distributed greedy algorithm starts by having each agent r_i compute the weights of

²Note that for a minimization problem, the approximation factors are greater than 1.

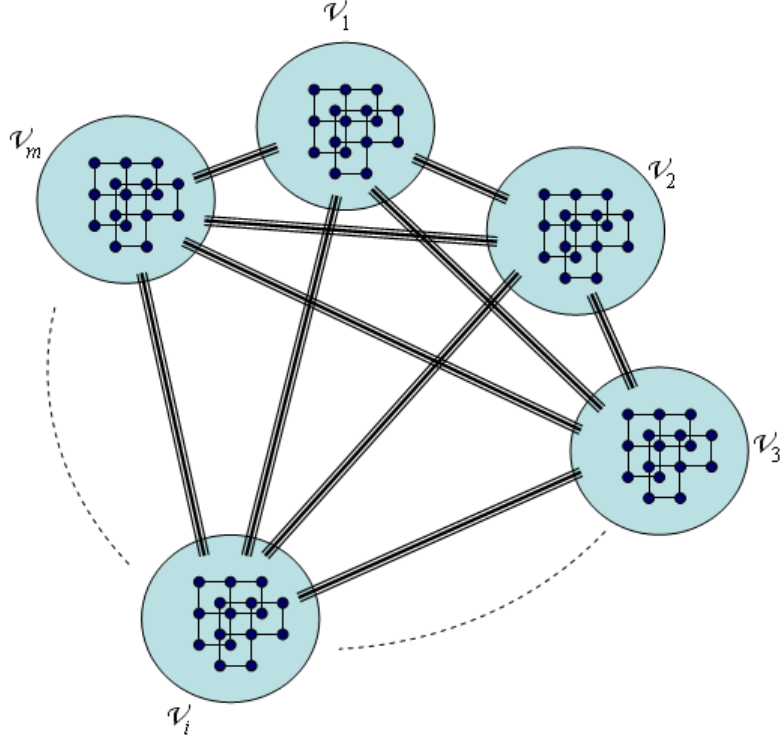


Figure 16: Dividing the set of graph vertices among agents.

all of the vertices in \mathcal{V}_i such that the weight of a vertex v is the sum of the weights of all incident edges including any common edges between \mathcal{V}_i and any other subset. r_i broadcasts the pair (\hat{v}_i, \hat{w}_i) containing the best local candidate vertex \hat{v}_i and its corresponding weight \hat{w}_i . After r_i has received similar broadcast messages from other agents, it compares its local maximum weight with the remote ones. If the local maximum weight \hat{w}_i is the highest, then \hat{v}_i is added to the solution set and removed from \mathcal{V}_i , and all incident edges will be removed including those that may be common between \mathcal{V}_i and any other subset. On the other hand, if the global best candidate vertex \hat{v} is different from \hat{v}_i , the agent will add \hat{v} to the solution. Then, it will check if there are any edges connecting any local vertex v_i and the best candidate vertex \hat{v} , in which case, it removes those edges, and it subtracts their weights from the weight of v_i . The previous scenario is repeated m times. Thus, the distributed greedy algorithm entails sending $O(m)$ messages and guarantees a solution that is at least $1 - e^{-1} \approx 63\%$ of the optimal solution and 84% of the best available centralized solution. At the end, each agent will have a copy of the solution. Agents then move to

the new locations. The way agents assign observation points among themselves is discussed in Section 6.3. Figure 17 shows a pseudo-code representation of the distributed greedy algorithm. Again, a clock symbol “🕒” placed next to a step number indicates that a timeout is used with the step to avoid waiting indefinitely on a message from a failed agent.

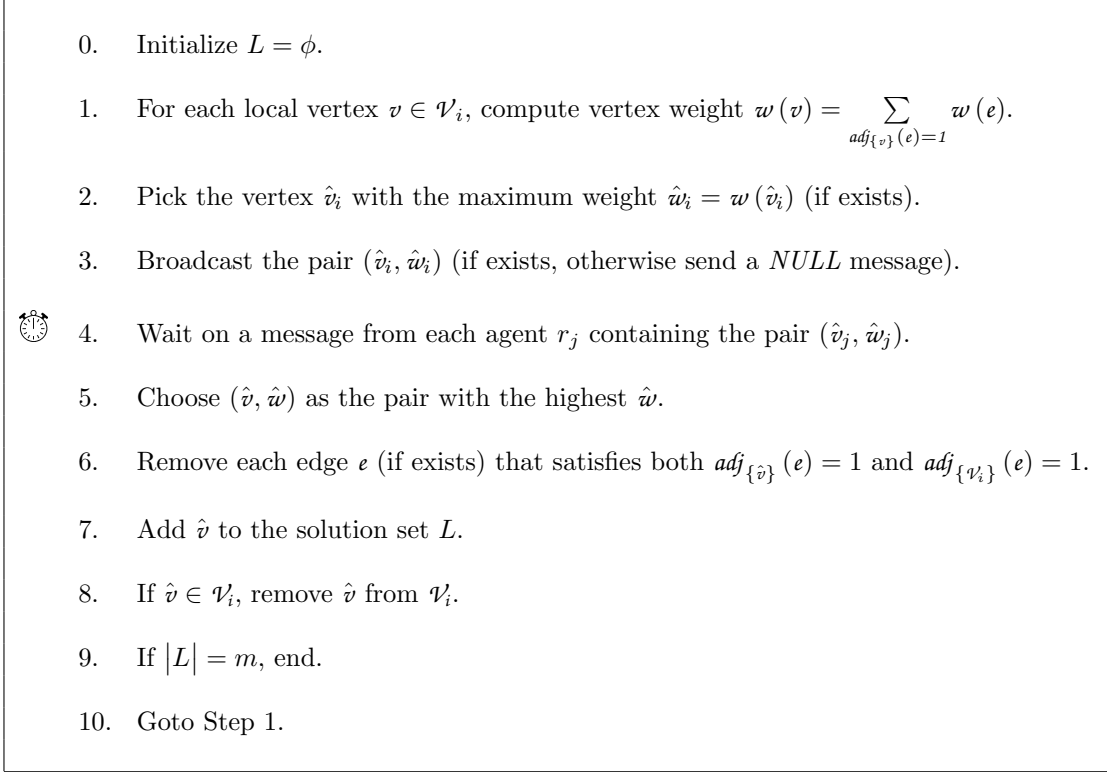


Figure 17: Pseudo-code representation of the distributed greedy algorithm.

6.1.3 Algorithm Complexity

In the following worst-case complexity analysis it is assumed that the set of vertices $|\mathcal{V}|$ are distributed evenly over agents, in which case, the biggest size of each subset \mathcal{V}_i is $\lceil \frac{|\mathcal{V}|}{m} \rceil$. The analysis also uses Δ as the maximum vertex degree, i.e., the maximum number of edges incident to one vertex in \mathcal{G} . Referring to Figure 17, the worst-case complexity of each step can be determined as follows:

- **Step 0:** $O(1)$.
- **Step 1:** $O\left(\Delta \lceil \frac{|\mathcal{V}|}{m} \rceil\right)$; assuming all vertices in \mathcal{V}_i have the maximum degree Δ .

- **Step 2:** $O\left(\left\lceil\frac{|\mathcal{V}|}{m}\right\rceil\right)$; one scan over \mathcal{V}_i determines the vertex with the highest weight.
- **Step 3:** $O(1)$; sending one message of constant size.
- **Step 4:** $O(m)$; receiving $m - 1$ messages.
- **Step 5:** $O(m)$; one scan over a list of m pairs.
- **Step 6:** $O(\Delta)$; worst case corresponds to $\hat{v} \in \mathcal{V}_i$ and \hat{v} has a maximum degree.
- **Step 7:** $O(1)$.
- **Step 8:** $O(1)$.
- **Step 9:** $O(1)$.
- **Step 10:** $O(1)$.

Steps 1 to 10 are executed for m rounds. The worst-case complexity for each round can be reduced to $O\left(m + \Delta \left\lceil\frac{|\mathcal{V}|}{m}\right\rceil\right)$. Thus, the total worst-case complexity for the m rounds is $O\left(m^2 + m\Delta \left\lceil\frac{|\mathcal{V}|}{m}\right\rceil\right)$. Note that the algorithm takes as an input a weighted graph. If the algorithm were to take the currently observed targets and their locations (expressed in edges), one pass over the set of observed targets can build the weighted graph. This pass incurs an extra $O(n)$ in the worst case, making the overall worst-case complexity of the algorithm

$$O\left(n + m^2 + m\Delta \left\lceil\frac{|\mathcal{V}|}{m}\right\rceil\right). \quad (24)$$

Since $\left\lceil\frac{|\mathcal{V}|}{m}\right\rceil < \frac{|\mathcal{V}|}{m} + 1$, making the substitution in Equation 24, the worst-case complexity becomes $O\left(n + m^2 + \Delta|\mathcal{V}| + m\Delta\right)$. In almost all cases, Δ (the maximum number of street segments incident to a street intersection) is 4. In rare situations, it may be 5 or 6. However, it cannot increase indefinitely. Taking this into account, Δ can be treated as a constant, yielding a final worst-case complexity of $(n + m^2 + |\mathcal{V}|)$, which scales well with the number of targets, the number of agents, and the graph size.

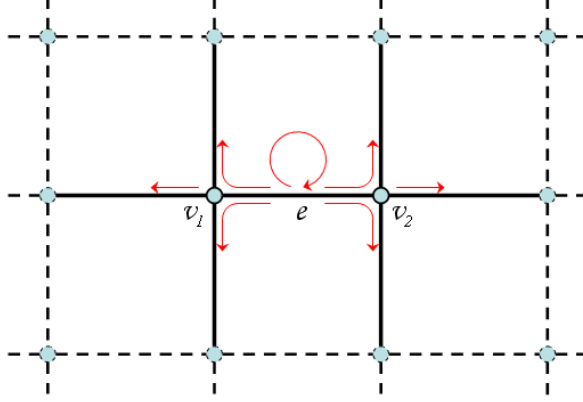


Figure 18: Target transitions defined on a graph.

6.2 Model-Based Algorithms

The only extra assumption that model-based algorithms make, over those made by the non-model-based algorithms, is that target transitions among street segments follow a stochastic model described by an M^{th} -order Markov chain as shown in Equation 18 in the previous chapter. The same equation is repeated here as Equation 25 for the reader's convenience. A target currently residing on an edge e connecting the vertices v_1 and v_2 may stay on the same edge e or move to any other edge incident to v_1 or v_2 as depicted in Figure 18.

$$\begin{aligned} \mathbf{X}(t) &= \sum_{i=1}^M \lambda_i \mathbf{Q}_i \mathbf{X}(t-i); \\ \sum_{i=1}^M \lambda_i &= 1, \end{aligned} \quad (25)$$

where $\mathbf{X}(t)$ here is a vector representing the probability distribution of target location over the street segments at time t . Agents can use the model to predict the target locations at future time instants as probability distributions using

$$\hat{\mathbf{X}}_j(t+h) = \begin{cases} \sum_{i=1}^M \lambda_i \mathbf{Q}_i \hat{\mathbf{X}}_j(t+h-i) & \text{if } h > M \\ \sum_{i=1}^{h-1} \lambda_i \mathbf{Q}_i \hat{\mathbf{X}}_j(t+h-i) + \sum_{i=h}^M \lambda_i \mathbf{Q}_i \mathbf{\Xi}_j(t+h-i) & \text{if } h \leq M \end{cases} \quad (26)$$

where $\mathbf{\Xi}$ is a vector describing the best estimate of the location at past time instants where fused sensor data is already available. Equation 26 handles two different cases: $h > M$ and $h \leq M$ (see Figure 11). Refer to the previous chapter for more details.

The weight function of Equation 22 can then be replaced by the *probable edge outcome*

(in analogy to the probable cell outcome defined in the previous chapter) for a future time horizon H

$$w(e, H) = \sum_{o_j \in O} u_j \left(\sum_{h=1}^H v_h \left(\hat{\mathbf{X}}_j(t+h) \cdot \mathbf{Y}_e \right) \right);$$

$$\sum_{h=1}^H v_h = 1, \tag{27}$$

where v_h describes the contribution of the future location prediction at time $t+h$ to the placement decision, and \mathbf{Y}_e is a binary vector with all zeros and a one at the location corresponding to the edge e . The equation sums up the *probability* \times *utility* quantities over all observed targets and over the set of time horizon H .

The way weights are assigned to edges is the only difference between the model-based and the non-model-based algorithm. Thus, the model-based algorithm is the same distributed greedy algorithm described above except that the equation in Step 1 is replaced by Equation 27. It is expected that the new edge weights will improve the overall coverage since they account for the future target transitions, whereas a non-model-based algorithm assumes that targets stay in their current locations at least until the next decision is made. The model estimation approach described in Section 5.3 can still be applied to estimate the parameters of the high-order Markov chain describing target transitions among street segments.

6.3 Optimizing Power Consumption

At the end of each execution of any of the algorithms described in the previous sections, each agent will have a copy of the solution L containing the best m OPs that agents have to move to before the next sampling period. A straightforward method to assign OPs to agents is to have each agent r_i move to OP l_i . The problem with this method is that it is not necessarily the optimum assignment in terms of power consumption. Hence, at the end of each execution, agents search for the optimum assignment that causes the least average agent motion. This in turn improves the average agent speed over the whole mission and hence the overall consumption of power used for agent navigation.

After finding the best m OPs, agents are charged with solving the following assignment problem: Given:

- A set of m agents $\{r_1, r_2, \dots, r_m\}$
- A set of m OPs $\{l_1, l_2, \dots, l_m\}$
- A cost matrix \mathbf{C} such that c_{ij} represents the cost of moving agent r_i from its current location to l_j ,

the goal is to find an assignment set $\Theta = \{(i, j) \mid i, j = 1, 2, \dots, m\}$ with a complete assignment (whereby l_j is assigned to r_i) that minimizes $\sum_{(i,j) \in \Theta} c_{ij}$.

The cost of moving an agent from its current location to the new OP can be expressed in terms of the distance to be traveled by an agent. In the case of a high altitude endurance UAV agent, Euclidean distance can be used since the agent can fly to the next OP in a straight line over buildings. On the other hand, the cost for a mini-UAV that flies at a low altitude is the shortest street path from its current location to the new OP. The shortest paths can be computed for every pair of vertices in a graph using Dijkstra's algorithm in polynomial time. Note that, generally speaking, since the graph \mathcal{G} is static in terms of its vertices' locations and edge adjacency, the shortest paths between every pair of vertices can be computed once beforehand and saved in a lookup table. Thus, retrieving such costs can be done in constant time.

The centralized Hungarian algorithm [23] finds optimal solutions for the assignment problem in polynomial time. A distributed algorithm called the *auction algorithm* [12] can solve the assignment problem described above with a (sequential) worst-case complexity of $O\left(\frac{m^3 C}{\epsilon}\right)$, where C is the maximum cost value. The algorithm optimality depends on a control parameter ϵ called *complementary slackness*. An optimal solution is obtainable if $\epsilon < \frac{1}{m}$. Since the auction algorithm starts with a maximization problem, the assignment problem described above can be reformulated as a maximization problem by updating the cost matrix according to

$$\tilde{c}_{ij} = \max_k \{c_{ik}\} - c_{ij}, \quad (28)$$

that is to subtract each element from the maximum element on the same row. The new cost matrix $\tilde{\mathbf{C}}$ expresses the distance saved by an agent by moving to an OP other than the furthest one. The problem then seeks an assignment set $\Theta = \{(i, j) \mid i, j = 1, 2, \dots, m\}$ with a complete assignment (whereby l_j is assigned to r_i) that maximizes $\sum_{(i,j) \in \Theta} \tilde{c}_{ij}$. In the rest of the section, it is shown how the auction algorithm can be adapted to the agent-OP assignment problem.

For each OP l_j , the auction algorithm defines a price p_j . Therefore, each agent maintains a vector \mathbf{P} of the prices associated with OPs. The price vector can be initialized to the zero vector. Then, it gets updated as the algorithm is executed. The algorithm then defines the *profit margin* of an agent r_i as

$$\pi_i = \max_{l_j \in L} \{\tilde{c}_{ij} - p_j\}, \quad (29)$$

which can be thought of as the benefit received by the agent r_i if it is assigned l_j . Based on these quantities, the algorithm goes through two phases each iteration. Figure 19 shows a pseudo-code representation for each iteration. The algorithm is guaranteed to terminate in a finite number of steps resulting in a feasible assignment, which is optimal if $\epsilon < \frac{1}{m}$. The auction algorithm can also run in a fully asynchronous (chaotic) fashion and still terminate in a finite number of steps [12].

6.4 Dealing with Evasive Targets

The distributed algorithms developed in this chapter and the previous chapter generally deal with non-evasive targets. In other words, the actions taken by agents are assumed to have no effect on the behavior of targets. This can be the case if targets cannot sense the agents, if targets have no knowledge they are being tracked by agents, or if targets have no reason to avoid agents. However, in many real-life situations, and especially in military applications, targets may attempt to run away from agents. In such situations, the decision making process on the agents' side becomes much harder because an action taken at any time step will affect the progress of the scenario at later steps. In consequence, agents are required to be more careful about their placement decisions. They should account for the effect of the decisions they make at each step on the future steps.

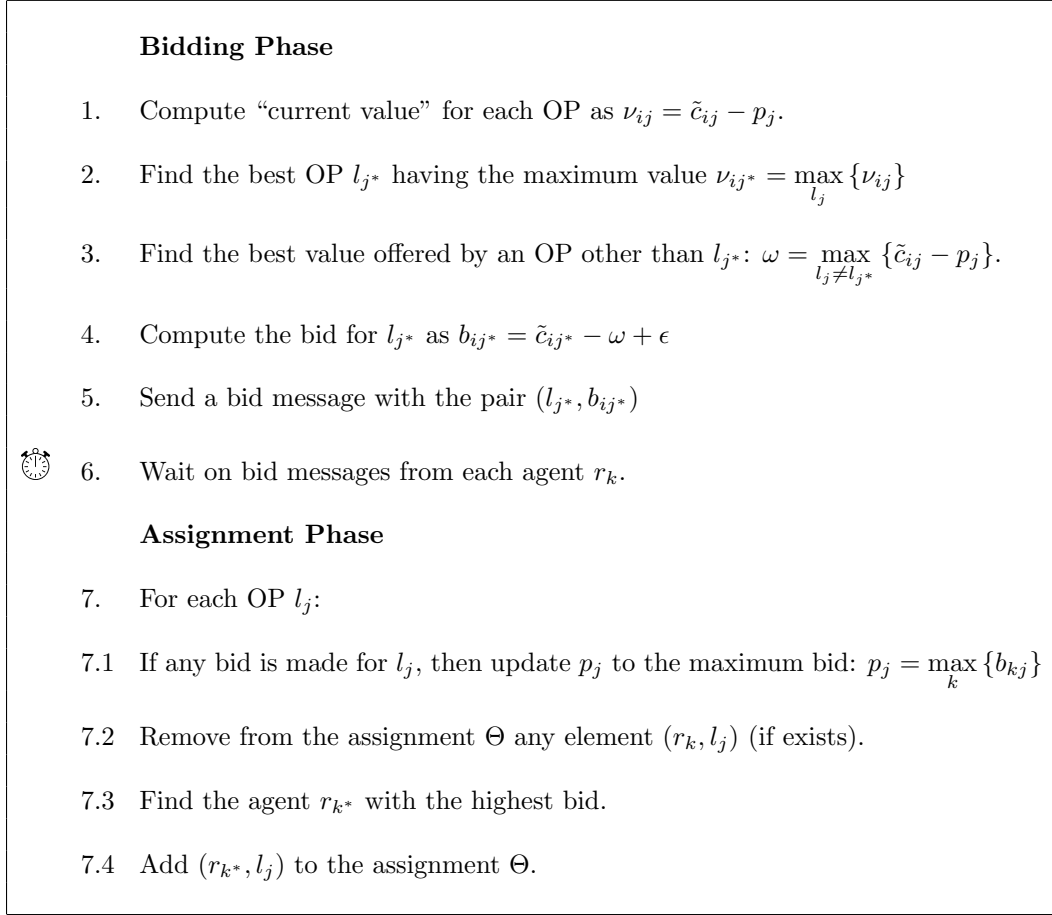


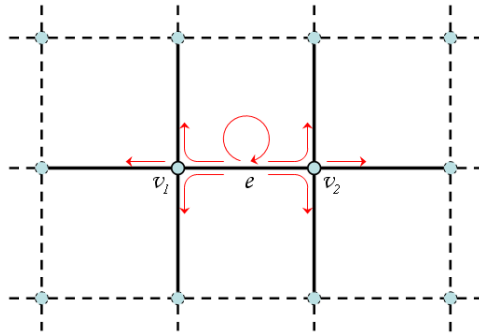
Figure 19: Pseudo-code representation of a single iteration of the auction algorithm.

6.4.1 Evasion Model

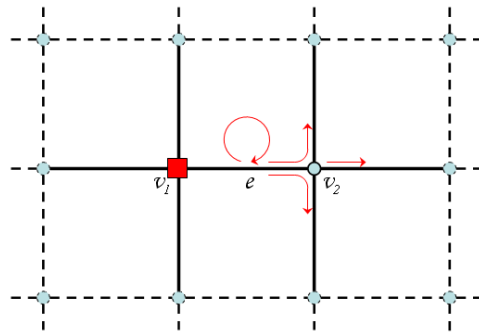
The M^{th} -order Markov chain (used to describe non-evasive target motion) can be modified to accommodate target evasion. The new evasive motion model can be described as a controlled Markov chain using agents’ locations as control parameters. Let \mathbf{V} be a vector representing the agents’ locations in terms of the vertices they are placed on. The high-order controlled Markov chain can be expressed as

$$\begin{aligned} \mathbf{X}(t) &= \sum_{i=1}^M \lambda_i \mathbf{Q}_i(\mathbf{V}) \mathbf{X}(t-i); \\ \sum_{i=1}^M \lambda_i &= 1. \end{aligned} \tag{30}$$

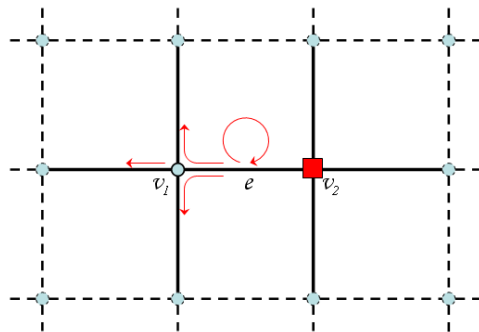
where the transition matrices \mathbf{Q}_i ’s describing the target motion are now functions of agents’ locations \mathbf{V} . Recall that each column in \mathbf{Q}_i represents a probability distribution of the next



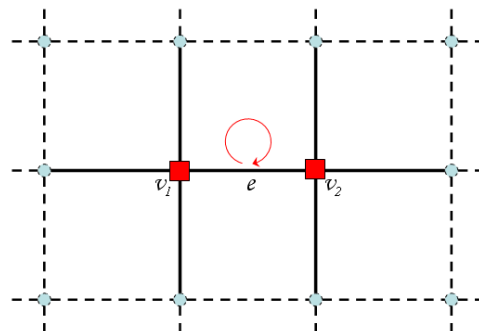
(a) e not observed.



(b) e observed from v_1 only.



(c) e observed from v_2 only.



(d) e observed from both v_1 and v_2 .

Figure 20: Target evasion model.

target location given that the target is on the graph edge represented by that column. In the evasive case, the probabilities in each column (representing an edge e connecting the vertices v_1 and v_2) are updated independently from other columns according to the rules shown below. Assume a target is currently on an edge e , all the relevant scenarios that may happen can be summarized as follows:

- If there is no agent at either v_1 or v_2 to observe e , the target will follow the non-evasive model (Figure 20(a)).
- If e is observed by an agent at v_1 , and there is no agent at v_2 , the target will only move to the edges incident to v_2 (Figure 20(b)).
- If e is observed by an agent at v_2 , and there is no agent at v_1 , the target will only move to the edges incident to v_1 (Figure 20(c)).
- If e is observed by two agents at both v_1 and v_2 , the target will stay trapped on the same edge (Figure 20(d)).

Thus, the column corresponding to e in transition matrix is reshaped according to the agent placement. The probabilities associated with transitions to “avoided” edges are set to zero. The probability corresponding to staying on the same edge is scaled down by $1 - \rho$, where $0 \leq \rho \leq 1$ is called the *evasiveness degree* because it represents how much the target is trying to escape the current monitored street segment. The probabilities associated with the other edges representing the “escape routes” are scaled up provided that the whole column adds up to 1.

6.4.2 Dynamic Programming Approach

Since the environment (represented by targets) responds to the actions taken by agents in the new set up, the problem can be modeled as a Markov Decision Process (MDP) [35] defined by the tuple

$$(\mathcal{X}, \mathcal{A}, \mathcal{T}, \mathcal{R}), \tag{31}$$

where

\mathcal{X} is the state space represented by all possible combinations of target locations,
 \mathcal{A} is the action space represented by all possible movements that agents can make,
 \mathcal{T} is the transition function described according Equation 30, and
 $\mathcal{R} : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function defined as the sum of the utility values of all observed targets as a result of placing the agent according to a certain action.

The nature of the new problem dealing with evasive targets is thus more amenable to dynamic programming (DP). The main difficulty in applying the DP approach in its pure form is the curse of dimensionality associated with the exponential increase in the size of the state space $|\mathcal{X}|$ and the size of the action space $|\mathcal{A}|$ as the number of agents and the number of targets increase. The curse of dimensionality has two direct impacts on the applicability of the DP approach. First, the space complexity of the representations of the state space may make it hard or even impossible to store all states using today’s technology. Second, large action space sizes increase the time needed to find an optimal solution to the MDP since the search time increases with the action space size. As a result, solving the problem using a straightforward approach such as *value iteration* or *policy iteration* may not be feasible for the application at hand. Moreover, using value iteration or policy iteration to find optimum solutions entails re-solving the whole problem offline every time an agent or a target is added. Therefore, the choice is made to apply an approach similar to real-time dynamic programming (RTDP) [6] as described below.

6.4.2.1 State and Action Representation

The state is represented by the weights of the graph edges defined by either Equation 22 or Equation 27. This makes the size of the state space $|\mathcal{X}|$ a function of the size of the region of interest not the number of targets. The weight of each edge is quantized to ℓ levels (from 0 to $\ell - 1$). Figure 21 shows an example graph with the system state represented by the probable edge outcomes quantized into four levels. Thus the size of the state space is of the order $O(\ell^{|\mathcal{E}|})$, which could still be a very large figure. However, it is likely that only a small subspace of this large space is visited in real situations. Hence, a sparse representation is

used for tables that are defined on \mathcal{X} such as the value function as will be described later in this section. Defining the state as a function of edge weights helps reducing the MDP to a deterministic one. In other words, if the graph is in a state \mathcal{x}_i , and agents took a certain action a_j , the new state of the graph can be represented in terms of the probable edge outcomes defined in Equation 27.

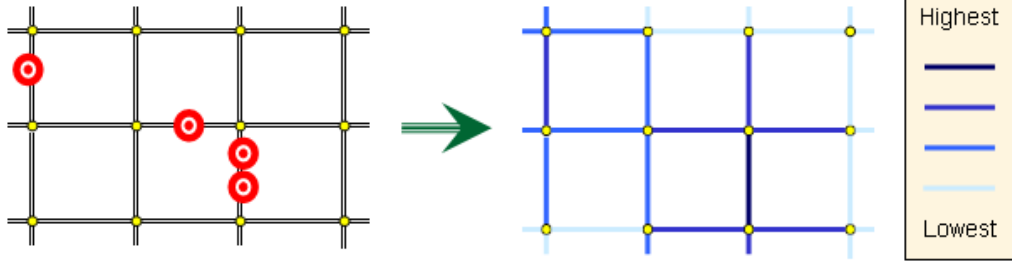


Figure 21: System state represented by quantized probable edge outcomes.

The action space is represented by all possible combinations defined on the vertex set \mathcal{V} . Each combination represents a possible agent placement over OPs.

6.4.2.2 RTDP Algorithm

The RTDP algorithm described here approximates the value iteration algorithm defined on an infinite horizon whereby the “value” V (or the profit-to-go) of each state can be optimized according to the iterative equation

$$V(\mathcal{x}_i) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(\mathcal{x}_i, a) + \delta \sum_{\mathcal{x}_j \in \mathcal{X}} P_a(\mathcal{x}_j | \mathcal{x}_i) V(\mathcal{x}_j) \right\} = \max_{a \in \mathcal{A}} Q(\mathcal{x}_i, a), \quad (32)$$

where $P_a(\mathcal{x}_j | \mathcal{x}_i)$ is the transition probability from \mathcal{x}_i to \mathcal{x}_j upon taking action a , and δ is a *discount factor* such that $0 < \delta < 1$ to discount the future reward at a geometric rate and thus prevent the values from increasing indefinitely. For every value function V , there is an associated policy defined by

$$\Pi(\mathcal{x}_i) = \arg \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(\mathcal{x}_i, a) + \delta \sum_{\mathcal{x}_j \in \mathcal{X}} P_a(\mathcal{x}_j | \mathcal{x}_i) V(\mathcal{x}_j) \right\} = \arg \max_{a \in \mathcal{A}} Q(\mathcal{x}_i, a), \quad (33)$$

that is to take the action that maximizes the total value. If the value function V is an optimal value function V^* , the policy Π corresponds to an optimal policy Π^* .

The algorithm maintains two main data structures: the value table V and the policy table Π , both defined over the state space \mathcal{X} . Sparse representation is used to overcome space limitations. Both tables can be initialized arbitrarily. The algorithm chooses an action according to Equation 33, but instead of considering all the actions $a \in \mathcal{A}$, it only considers a smaller set of actions $\{a_s, a_r\} \cup \{a_{h_i} | i = 1, 2, \dots, J\}$, where $a_s = \Pi(\chi)$ is the stored action, a_r is an action chosen randomly, and $\{a_{h_i} | i = 1, 2, \dots, J\}$ is a set of J heuristic actions. After choosing the best action according to Equation 33, the policy table is updated to the best action among the $J + 2$ actions, and the value table is updated accordingly as well. The algorithm is guaranteed to reach an optimal policy provided that the all the states are visited infinitely often [6]. The value and the policy tables represent the agents' knowledge (acquired so far) of dealing with evasive targets, and thus, they belong to the knowledge level shown in Figure 5. The algorithm can be implemented in a distributed fashion. Figure 22 shows a pseudo-code representation of the distributed algorithm running on agent r_i after the state of the graph χ (represented in terms of edge weights) has been determined.

Each step of the distributed algorithm running on an agent r_i is explained as follows:

- **Step 1:** The agent generates a random number α_{r_i} taking a value between 1 and $|\mathcal{V}|$ inclusive. It then broadcasts α_{r_i} .
- **Step 2:** The agent waits on messages with random numbers generated by other agents.
- **Step 3:** If some of the random numbers α_{r_j} and α_{r_k} are similar, the agent replaces the latter with the first available number scanning from 1 through $|\mathcal{V}|$.
- **Step 4:** The agent forms the random action as as the tuple generated by concatenating the random numbers generated by agents.

The purpose of steps 1 to 4 is to generate the random action in a distributed fashion. At the end of executing the four steps, each agent will have a consistent copy of the random action. The purpose of the random action is to explore the action space.



1. Generate and broadcast a random OP number α_{r_i} .
-  2. Wait for a message from each other agent r_j with a random number α_{r_j} .
3. For every similar pair (α_1, α_2) , replace α_2 by the first available OP number.
4. Form the random action as the tuple $\mathbf{a}_r = (\alpha_{r_1}, \dots, \alpha_{r_m})$.
5. Look up the stored action from the policy table $\mathbf{a}_s = \Pi(\chi)$.
6. Compute the heuristic action tuples $\mathbf{a}_h^{(1)}, \dots, \mathbf{a}_h^{(J)}$.
7. Evaluate and broadcast the local value of each action tuple $q_i(\chi, \mathbf{a}_r), q_i(\chi, \mathbf{a}_s), q_i(\chi, \mathbf{a}_h^{(1)}), \dots, q_i(\chi, \mathbf{a}_h^{(J)})$.
-  8. Wait on a message from each agent r_j containing its evaluated local values q_j 's.
9. Evaluate the global values Q 's from the local values q_κ 's.
10. Choose and execute the action tuple \mathbf{a}^* with the best global value Q^* .
11. Update the value table $V(\chi) = Q^*$.
12. Update the policy table $\Pi(\chi) = \mathbf{a}^*$.

Figure 22: Pseudo-code representation of the RTDP algorithm.

- **Step 5:** The agent looks up and extracts the action corresponding to the current state χ in the action table.
- **Step 6:** A number of heuristics are applied to the current state χ , and the corresponding heuristic actions are computed.
- **Step 7:** For each of the random action, the stored action, and the heuristic actions, the agent evaluates its local values as its local immediate reward plus its discounted (multiplied by δ) local future reward. The immediate reward is the sum of the weights of those edges incident to the vertex whose number is the i^{th} component of the action tuple. The future reward is the sum of the probable cell outcomes after applying the evasion model assuming agents will move according to the action tuple under consideration.

- **Step 8:** Wait for other agents to send their local values for each action to be evaluated.
- **Step 9:** The global value of each action is computed by adding up all the local values. If a certain action requires that an edge be observed from both ends, the value v accrued by monitoring the edge is subtracted once. Note that each of the two agents would have included v in its local value.
- **Step 10:** The action with the best global value is chosen and executed by all agents.
- **Step 11:** The value table is updated at the entry corresponding to the current state χ to the best global value found in the last step.
- **Step 12:** The policy table is updated at the entry corresponding to the current state χ to action corresponding to the best global value.

According to the RTDP-based algorithm described above, agents evaluate actions as if each agent r_i is going to move to the street intersection encoded in the i^{th} component of the action tuple. However, to actually move the agents to the new locations, agents execute the auction algorithm described in Section 6.3 to optimize the power consumed for navigation purposes.

CHAPTER VII

PERFORMANCE EVALUATION

In this chapter, the performance of the distributed agent placement approach is evaluated through sets of simulation experiments inspired from real-life scenarios. The chapter is divided into two sections, each of which is dedicated to evaluating the performance of the distributed approach in a different type of urban terrain.

7.1 Approach Evaluation for Type-I Urban Zones

7.1.1 Non-Model-Based Algorithms: CEP

This section compares the performance of the CEP algorithm against A-CMOMMT [31] since it is referred to as the “best-known human-designed policy” for the CMOMMT problem [37]. Hence, the simulation baseline used is similar to the one used in [33] to evaluate A-CMOMMT. A-CMOMMT uses virtual forces to attract an agent to targets and repel them from other agents. The forces are scaled according to certain functions that are defined based on the distance between an agent and targets and other agents.

Targets move according to a random linear pattern in which each target moves in a straight line and changes direction with a probability of 5%. The direction of motion first has a positive x component and any y component until the right boundary of the ROI is reached. After that, it has a negative x component until the left boundary is reached, and so on. The y component can be positive, zero, or negative at any time. The maximum target speed is 150 units per second, and the maximum agent speed is 200 units per second. Since both CEP and A-CMOMMT are non-model-based algorithm, agents have no knowledge about the target motion pattern.

To illustrate the robustness of CEP, several variations are considered. Each variation differs from the main CEP algorithm in the frequency the algorithm is applied. For example, CEP(20%) differs from CEP in the fact that it is only applied every 5 sampling periods

instead of every sampling period. Figure 23 shows the performance of CEP compared to A-CMOMMT (as defined in [33]) for a set of 5 agents and 20 targets, as the size of the region of interest (ROI) increases. The comparison is based on two metrics:

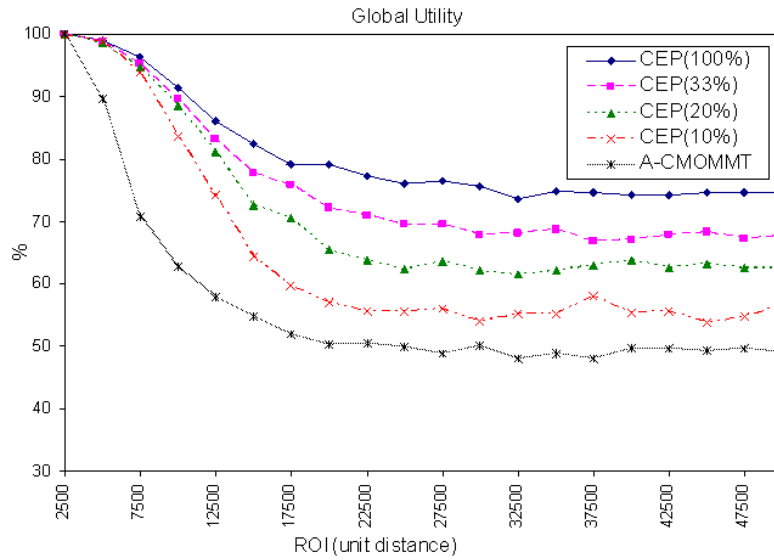
- the global utility defined in Equation 8, which is repeated here as Equation 34,

$$G = \frac{\sum_{t=0}^T \sum_{j=1}^n u_j \prod_{i=1}^m \gamma_{ij}(t)}{T \sum_{j=1}^n u_j} \quad (34)$$

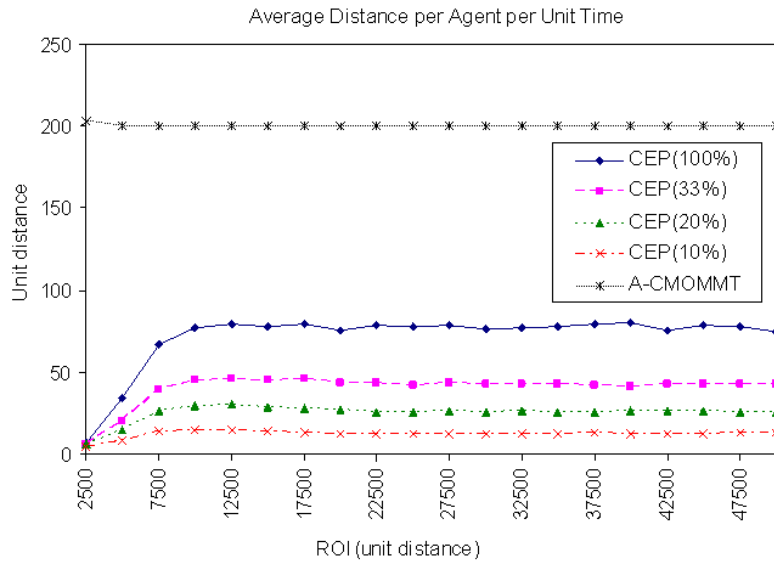
- the average distance traveled by an agent in a unit time.

The latter metric can be used as an indication of the power used by an agent for navigation purposes. Since A-CMOMMT assigns agents a fixed speed [33], the latter metric is expected to be constant. The set of experiments used to generate the plots shown in Figure 23 assumed that all targets have the same utility value since A-CMOMMT was originally designed to track targets of equal utility values. Thus, G , as defined in Equation 34, reduces to the metric A used to evaluate the A-CMOMMT performance in [33]. A single point on any of the plots corresponds to the average value computed over a set of 100 experiments. The figure clearly shows that all CEP variations consistently outperform A-CMOMMT in terms of both metrics discussed above. For instance, for large ROI sizes, the main CEP variation improves over A-CMOMMT by about 25% of the maximum achievable global utility and requires agents to move at an average speed less than half of that required by A-CMOMMT.

The same set of experiments is repeated for a range of target utility values that vary from 1 to 20. For this set of experiments, a variation of A-CMOMMT that we call WA-CMOMMT is tested, in which the local forces are scaled according to the targets utility as well as according to the scaling functions [33] used by A-CMOMMT. This makes agents favor targets with high utility values. Figure 24 shows the superiority of CEP over A-CMOMMT and WA-CMOMMT. At a first glance, it may seem surprising that A-CMOMMT exhibits better performance than WA-CMOMMT although the latter accounts for target utility values, while the former does not. The reason behind this outcome is that the latter may



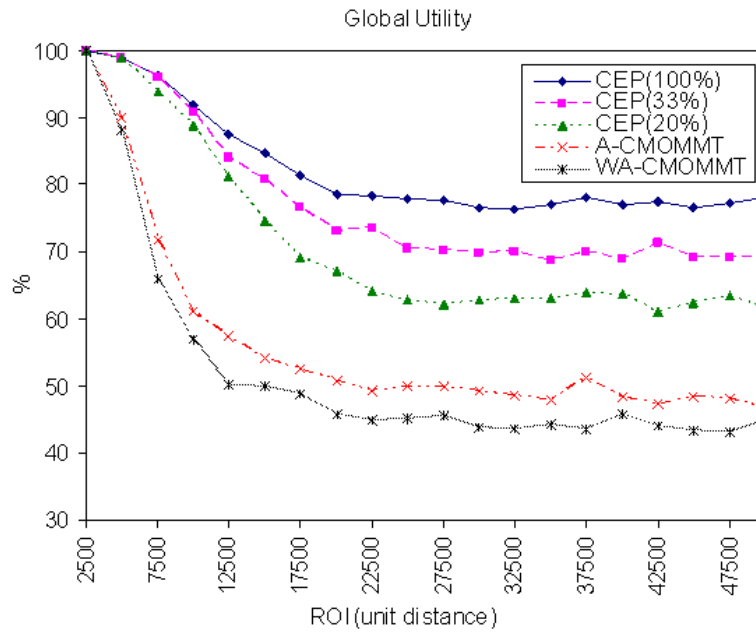
(a) Global utility vs. ROI size.



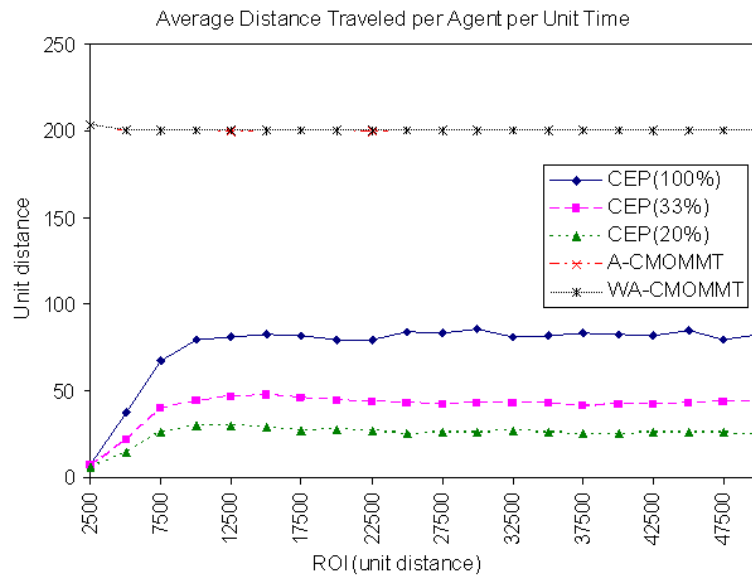
(b) Average distance traveled by an agent per unit time vs. ROI size.

Figure 23: CEP vs. A-CMOMMT when targets are of the same utility value.

cause two or more agents to get attracted to a single target with a very high utility. The attraction force to that target may be stronger than the repulsion force between the agents. This scenario leaves out other targets of less utility unobserved. If one agent were to observe that target of high utility while other agents tracked the other targets, a better global utility would be expected.



(a) Global utility vs. ROI size.



(b) Average distance traveled by an agent per unit time vs. ROI size.

Figure 24: CEP vs. A-CMOMMT when targets are of unequal utility values.

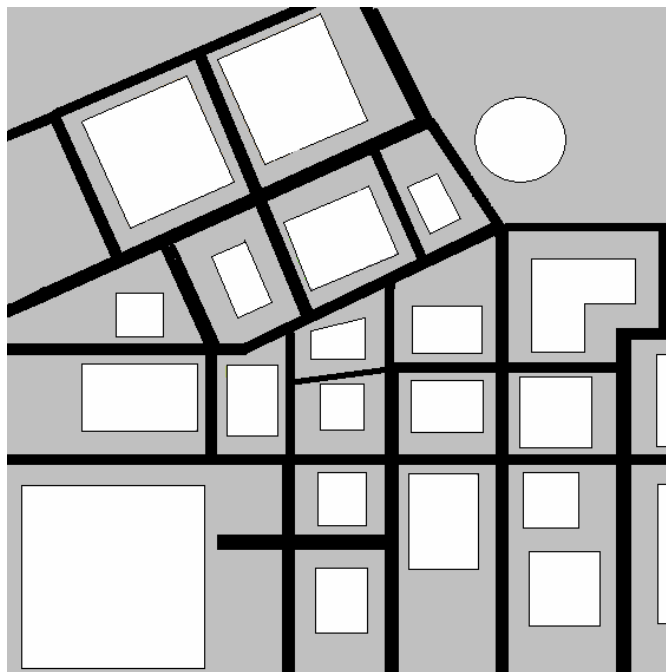


Figure 25: A virtual $1\text{ km} \times 1\text{ km}$ urban region (black lines represent roads, and white regions represent buildings/facilities)

7.1.2 Model-Based Algorithms: PCEP

In this section, we show through a set of simulation experiments that the model-based predictive algorithm PCEP further improves over CEP. In all experiments, we consider a virtual $1\text{ km} \times 1\text{ km}$ urban region (Figure 25). The region is divided into 20 meters by 20 meters cells. The stochastic model used here attempts to capture a navigation model for an urban target. Thus, the model transition probabilities are defined such that a target moving along a road will continue moving along the same road in the same direction with a high probability and may pull off the road with a much smaller probability. Transition probabilities for a target at a road intersection are defined such that a target has a chance to continue on the same road or turn right or left with different probabilities. A target in a cell that is not part of a road will remain in the same cell with a high probability and may leave the cell with a small probability.

The simulation is performed for a set of four similar aerial agents with a sensing range of 160 meters in each direction, a communication range of 400 meters, and a maximum speed

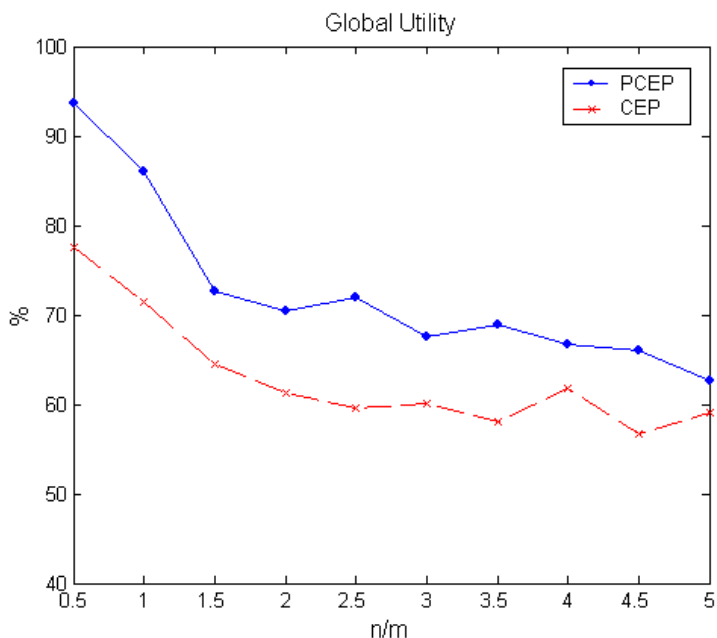


Figure 26: PCEP performance compared to CEP as the target-to-agent ratio increases.

of 576 kilometers per hour. Agents update their locations based on a thirty-second future time horizon. Figure 26 shows the global utility (aggregate coverage) obtained using CEP and PCEP as the number of targets vary from 2 to 18. Each point on the plot corresponds to the average value obtained from 20 experiments with each experiment running for 8 minutes. The figure shows that PCEP consistently outperforms CEP by about 5-15% of the maximum achievable global utility. This confirms the intuition that a model-based predictive approach is expected to deliver better performance than a non-model-based one since the former accounts for the future target transition, whereas the latter assumes that targets stay still until the next decision is made.

7.1.3 Online Model Estimation

7.1.3.1 Order Estimation: PACF Approach

The PACF approach for order estimation is tested for synthesized data. Synthesized sequences are generated according to a high-order Markov chain defined on the region of interest described in Section 7.1.2. The PACF is evaluated for each of the x and the y components separately. Figure 27 shows the absolute value of the PACF for each component

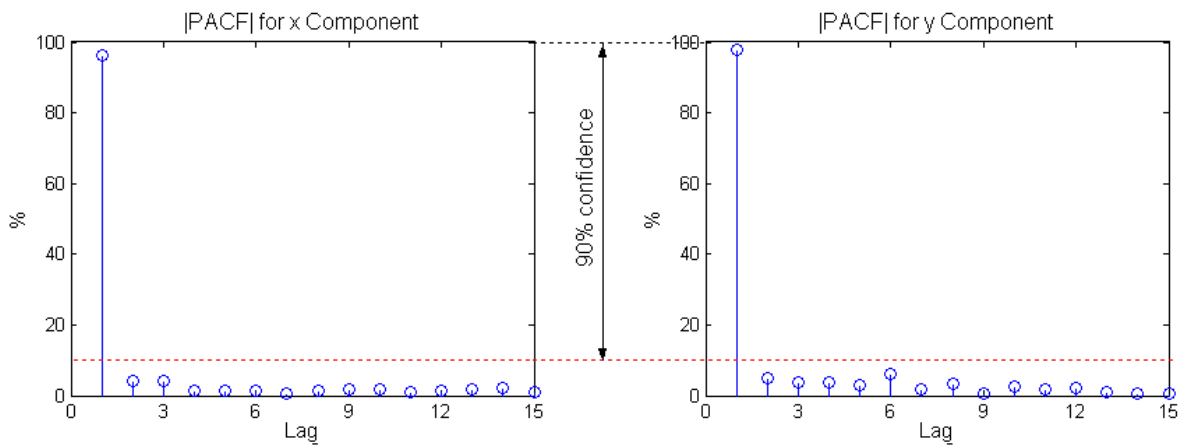
of a first-order, a third-order, and a sixth-order Markov chains ¹. The order is determined for a 90% confidence level. Thus, the order is determined as the last lag that has a value greater than or equal 10%. For the first-order and the third-order cases, the same order is obtained for both components according to the PACF approach (Figure 27(a) and Figure 27(b), respectively). For the sixth-order case (Figure 27(c)), the estimated order is 3 according to the x component and 6 according to the y component. Since the overall model order is estimated as the maximum of the two estimated orders obtained for each component separately, the PACF approach still estimated the correct order for the sixth-order case.

7.1.3.2 Parameter Estimation: Modified Berchtold's Algorithm

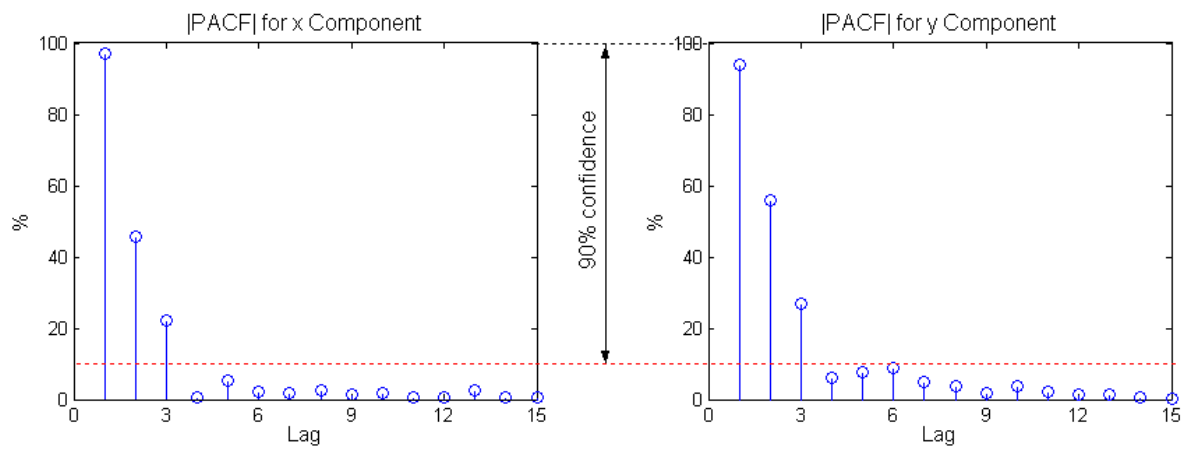
To evaluate the modified Berchtold's algorithm, synthesized sequences are generated according to a Markov chain with known parameters. The sequences are fed to the modified Berchtold's algorithm in an attempt estimate the parameters of the actual model. To evaluate the estimated model, both the estimated and the actual models are used to predict the future target location of a target currently at cell c using Equation 19 from Section 5.2.2. Recall that the model predicts a target location at a future time instant as a probability distribution defined over the cells of the ROI. The two predictions of both the actual and the estimated model are compared in terms of their mean values. The four-step ahead prediction error magnitude (in cells) between the estimated and the actual model is shown in Figure 28(a). The average error magnitude over the whole region of interest for the k -step ahead prediction is shown in Figure 28(b). An error bar is shown on the plot for one standard deviation above and below the curve. Since the curve shows the error magnitude which is always nonnegative, the error bars are clipped at zero. As we can see, the average prediction error for the ten-step ahead prediction is less than half a cell, meaning that the estimated model provides reasonably accurate predictions compared to the actual model.

Small prediction errors such as the ones shown above are expected to have only very slight effect on the performance of PCEP since the average error is very small (a fraction of a cell) compared to the agents' sensing range (tens of cells). Thus, the target is still

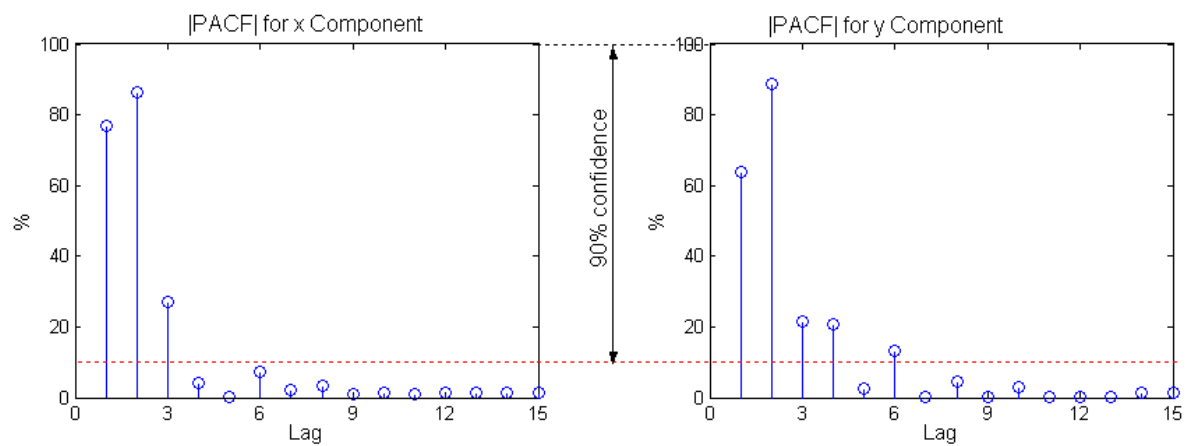
¹The PACF is not shown on any on the plots for zero lags, which is known to be always 1.



(a) First-order case.

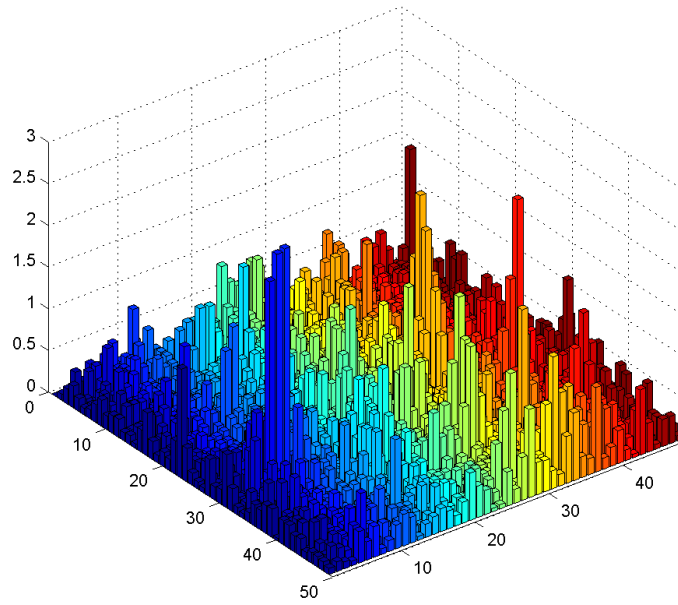


(b) Third-order case.

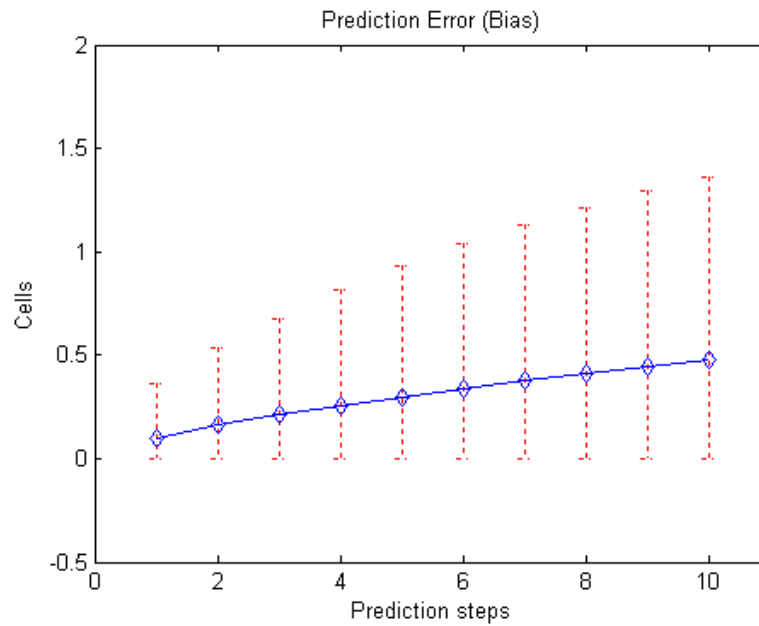


(c) Sixth-order case.

Figure 27: Estimating model order from the absolute value of the PACF (PACF is not shown for zero lags).



(a) Prediction error shown for each cell in the ROI.



(b) Average prediction error for different prediction steps.

Figure 28: Error in prediction due to using the estimated model.

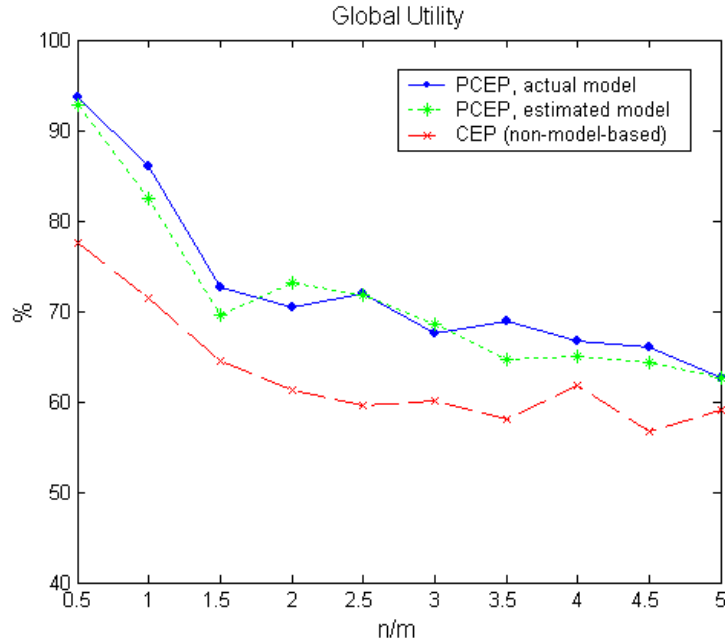
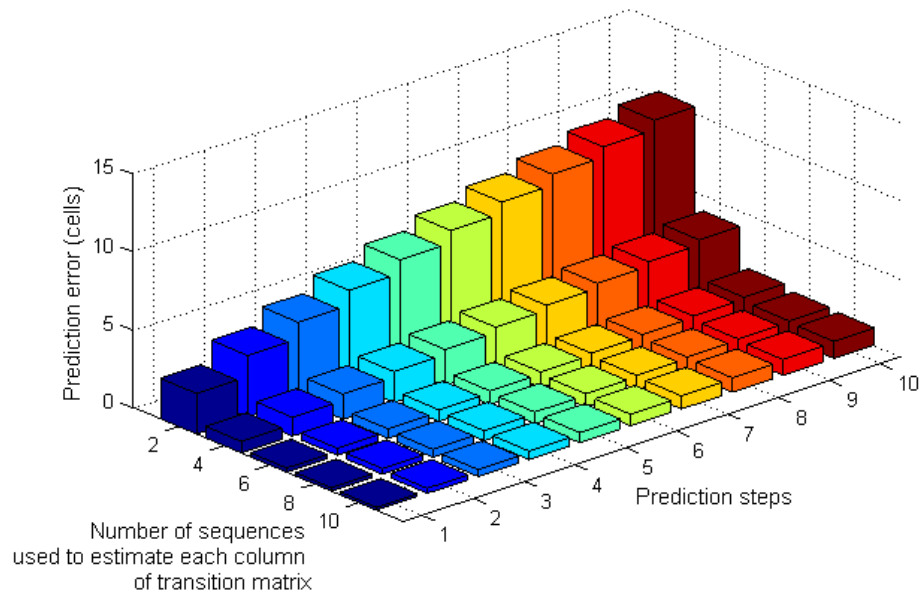


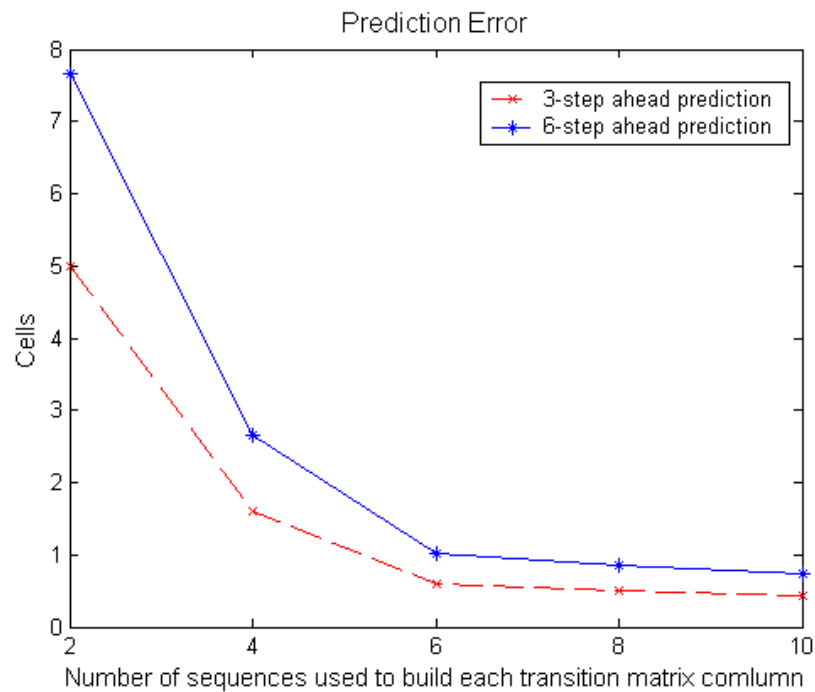
Figure 29: PCEP performance under actual and estimated models.

expected to fall in the agent’s sensing range under small prediction errors. To confirm this argument, the same simulation experiments described in Section 7.1.2 are repeated using the estimated model. Figure 29 shows that the performance of the estimated model closely follows that of the actual model with a very small degradation in performance. Even with this degradation, the model-based predictive algorithm, PCEP, still outperforms the non-model-based non-predictive CEP.

To study the effect of the data size used to estimate the model on the model-based algorithm performance, several models are estimated based on different data sets with different sizes. Data size is determined in terms of the number of fixed-size synthesized sequences it contains. Each of the estimated models are used to predict target locations as probability distributions. The prediction error (as defined earlier in this section) is averaged over all the whole ROI and is shown in Figure 30(a) for different models and for different future prediction steps. The figure clearly shows that as the data sized used in the model estimation increases, the prediction error consistently decreases for a fixed prediction step. Figure 30(b) shows how the prediction error varies as the size of the data used to estimate



(a) Prediction error shown for different prediction steps and for different models that were estimated based on different data sizes.



(b) Three and six-step ahead predictions as the data size used to estimate the underlying model increases.

Figure 30: Estimation data size effect on the estimated model quality.

the underlying model increases for the three-step and six-step ahead predictions. Hence, prediction error convergence can be used as a criterion to release the model. In other words, if using more data to estimate the model does not improve the prediction error by a huge margin, the model can be assumed accurate enough to be used by the model-based algorithms.

7.2 Approach Evaluation for Type-II Urban Zones

7.2.1 Overall Coverage: Distributed Greedy Algorithms

The algorithms developed in Chapter 6 are evaluated through simulation on a high-rise urban terrain zone whose street map is shown in Figure 31. The figure also shows OPs at street intersections and a few excessive OPs added to split long street segments into smaller segments. A first-order Markov chain is defined to determine how targets move from a street segment to another within the urban terrain zone. The model-based (predictive) as well as the non-model-based (non-predictive) versions of the distributed greedy algorithm developed in Chapter 6 are evaluated in comparison with a simple (naive) greedy algorithm [19], a random dynamic policy, and a fixed policy where agents are placed statically at OPs that are close to the center of the ROI. The simple greedy algorithm differs from the iterative one in that it picks the m OPs in one shot. This is done by sorting the graph vertices according to their weights and picking the first m vertices. Thus, the simple greedy algorithm is expected to place agents in such a way that several edges are observed from both ends. Generally speaking, the simple greedy algorithm is expected to deliver less coverage since in many cases, targets with high utility values are unnecessarily monitored by two agents.

Figure 32 shows the simulation results of applying different algorithms to the scenario described above. It is clear that the predictive iterative greedy algorithm exhibits the best performance among all of the algorithms under evaluation. The non-predictive version of the same algorithm, however, delivered worse performance than the two versions of the simple greedy algorithm, which may seem to be counterintuitive. Let us first compare the non-model-based (non-predictive) version of both algorithms. Note that because both algorithms assume that targets stay in their current locations (which is not the case), any

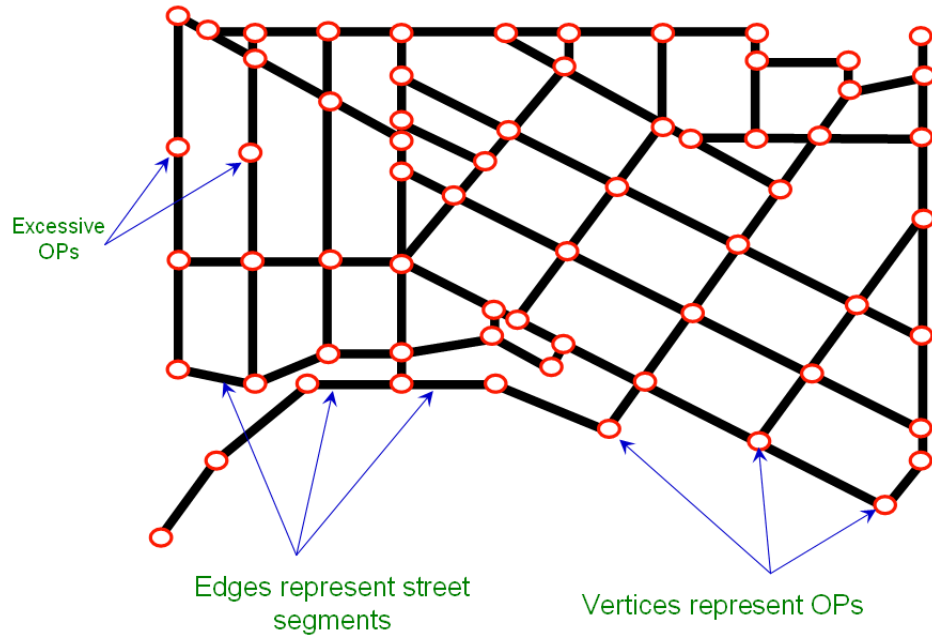


Figure 31: A graph representing Type-II urban terrain zone.

target that escapes the surveillance at a certain point may stay out of sight until it is “run into” by an agent at a later step. For that reason, the simple non-predictive greedy algorithm outperforms its iterative counterpart. It places an agent at each end of an edge containing one or more important targets, having a good grasp on those targets. Even if a target escapes in any of the two directions, the edges it can move to will still be monitored thanks to the two agents placed at the ends of occupied edge. The iterative greedy algorithm, on the other hand, usually results in placements where it is unlikely that an edge is monitored from both ends, increasing the chances that targets can run away. Now, we see the value of using a model to predict future target locations. The predictive version of the iterative algorithm shares with the non-predictive version the fact that it results in placements where it is unlikely that an important edge is monitored from both ends. However, it predicts where those targets will be in the future (up to some degree of certainty), and hence produces better placements that account for highly important targets and targets that are of less importance at the same time.

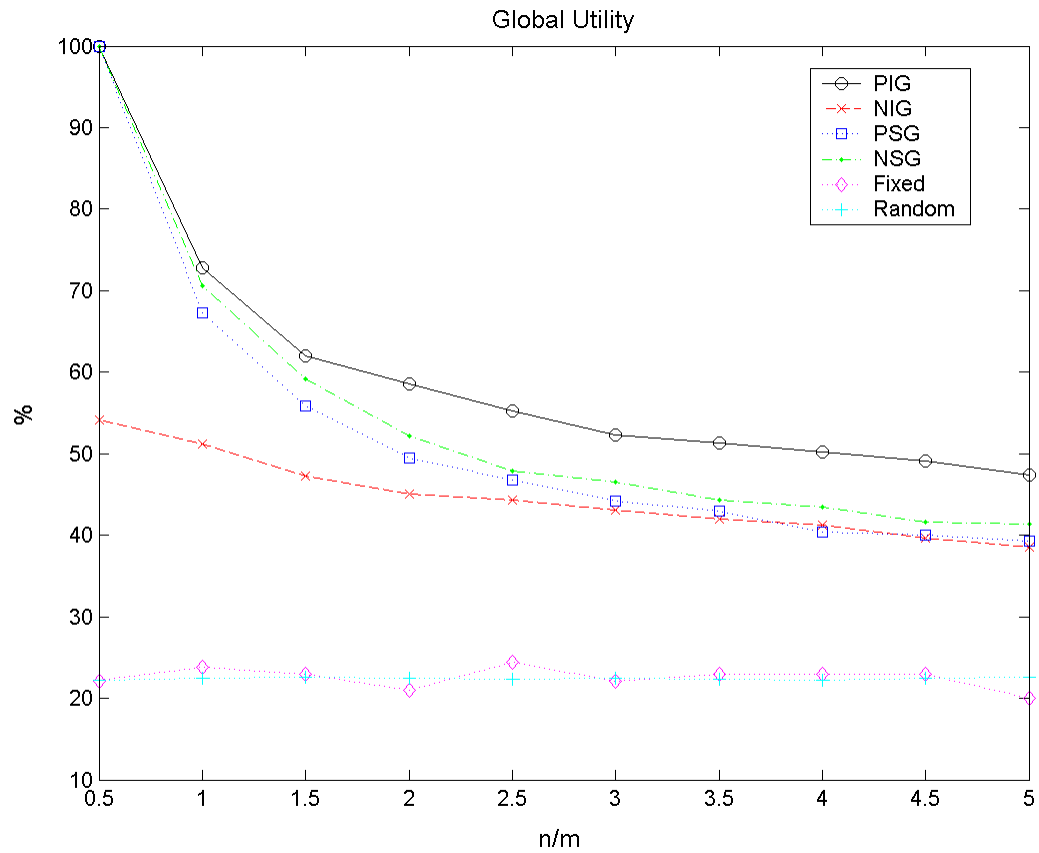


FIG: Predictive iterative greedy
 NIG: Non-predictive iterative greedy
 PSG: Predictive simple greedy
 NSG: Non-predictive simple greedy

Figure 32: Performance of different placement algorithms for Type-II urban zones.

7.2.2 Navigation Power Optimization: Auction Algorithm

Recall that the auction algorithms (see Section 6.3) attempts to improve the power consumption by choosing the optimal way that OPs are to be assigned to agents. In this section, the performance of the auction algorithm is evaluated in terms of the average distance traveled by an agent in a unit time. This gives an indication of the power consumed by agents for navigation purposes. The algorithm is evaluated for the same scenario defined on the urban terrain zone represented by the graph shown in Figure 31. Two types of agents are considered: mini-UAVs and high altitude endurance UAVs. For the mini-UAVs case, an agent moves from its current locations to its assigned OP by flying over the shortest street route. For regular UAVs flying at a high altitude, an agent moves from its current locations to its assigned OP in a straight line. The performance of the auction algorithm for each case is shown versus the unoptimized case where OPs are assigned to agents such that the i^{th} OP is assigned to the i^{th} agent. Figure 33 shows simulation results for the optimized and unoptimized cases. Note that the four cases deliver the same coverage; the only difference lies in how OPs are assigned to agents and how agents move to their assigned OPs. The figure indicates that using the distributed auction algorithm to solve the assignment problem achieves the same global utility while significantly reducing the average agent speed by about 65-75% of that required by the direct assignment. This in turn has a huge impact on the amount of power consumed by agents for navigation purposes.

7.2.3 Dealing with Evasive Targets: RTDP Approach

The RTDP algorithm presented in Section 6.4.2 is evaluated for two different scenarios. The first scenario corresponds to some cases where the optimal policy is obvious. For example, consider a scenario where the number of agents is at least double the number of evasive targets. According to the evasion model defined in Section 6.4.1, an evasive target can be trapped by placing an agent at each of the ends of the street segment where the target is located. Enforcing such policy guarantees obtaining the full utility of all targets (100% global utility). Notwithstanding that such scenario is trivial compared to the general case, it is used as a proof of concept since it is one of the scenarios where the optimal policy is easy

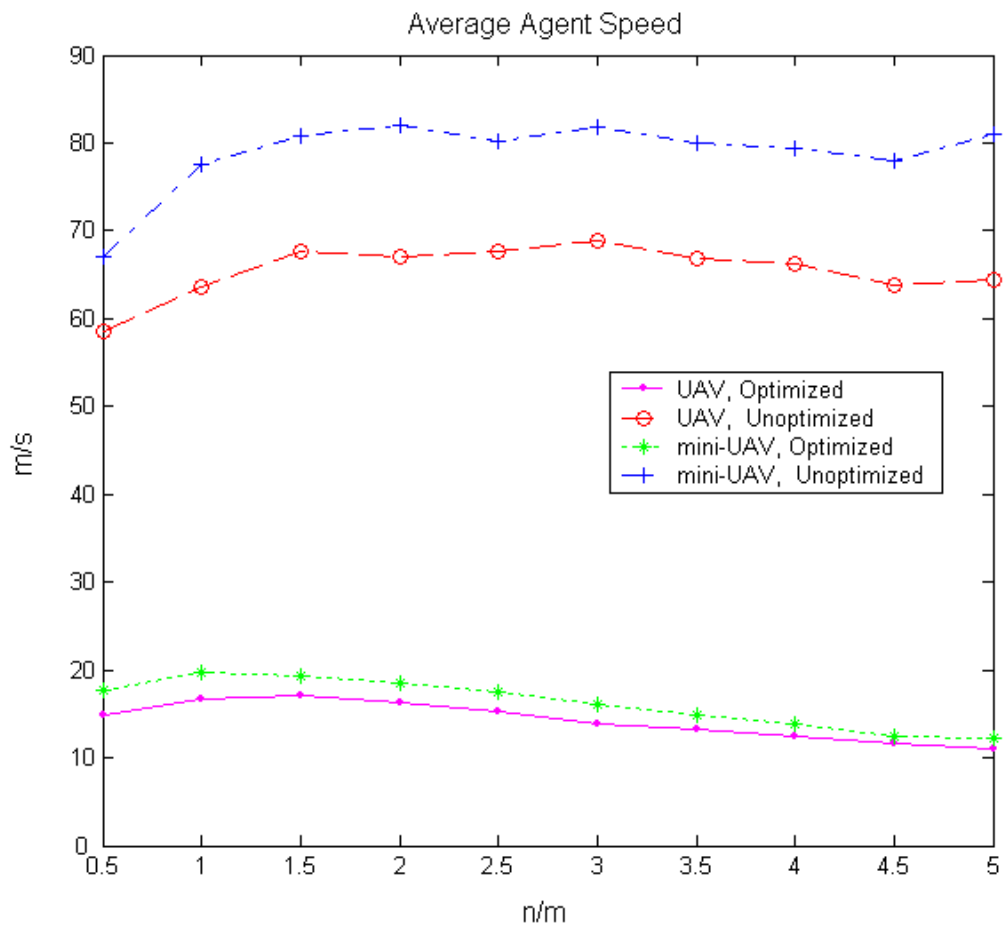
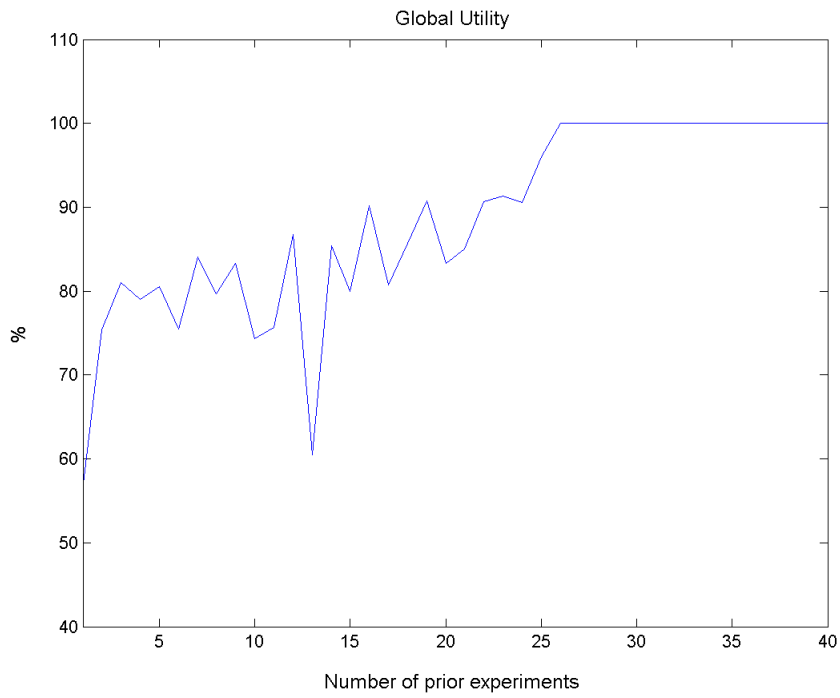


Figure 33: Optimizing average agent speed using the distributed auction algorithm.

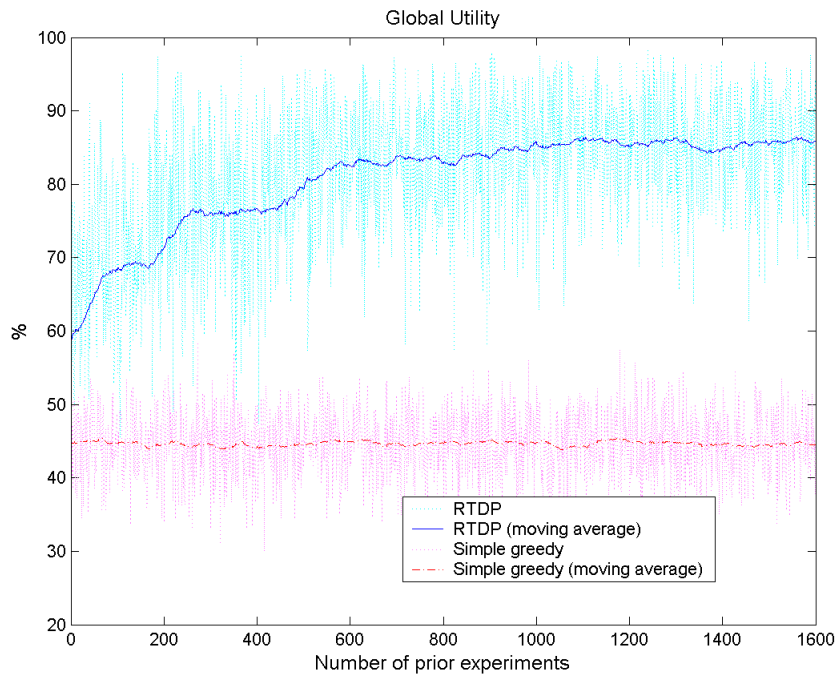
to comprehend without involving too much thinking. The RTDP algorithm is not provided with any heuristic in this case to see if it can learn the optimal policy on its own. It only chooses from a pool of two actions: the stored action and a random action (see Section 6.4.2 for more details). The algorithm was able to eventually learn the optimal policy on its own. Figure 34(a) shows the global utility accrued versus the number of experiments performed in advance to build the knowledgebase (represented in the value table and the action table).

The same set of experiments is repeated for a general case and the performance of the RTDP algorithm is compared to the simple greedy algorithm. While a “static” algorithm such as the simple greedy algorithm maintains the same level of performance, an intelligent dynamic algorithm such as the RTDP algorithm is able to improve its performance over time just by maintaining two data structures that store the knowledge gained so far from past experience. This is clearly shown in Figure 34(b).

Figure 35 shows a case where the RTDP algorithm enabled four agents to “contain” five evasive targets; no matter where any of the targets can go at this point, it will still be under the surveillance of at least one agent. According to the *trap* or *capture* event shown in Figure 20(d), none of the five targets is *captured* since none of the targets is observed from both ends of the edge where it currently exists. However, each of the targets is always observed by at least one agent, achieving 100% of the maximum achievable global utility. This example clearly illustrates the main difference between a pursuit-evasion game, where all targets need to be captured before the game terminates and the version of the agent placement problem defined in this thesis, where the primary goal is to keep as many targets as possible under surveillance for as long as possible.



(a) RTDP algorithm captures an optimal policy.



(b) RTDP algorithm improves its performance over time.

Figure 34: Performance of the RTDP algorithm.

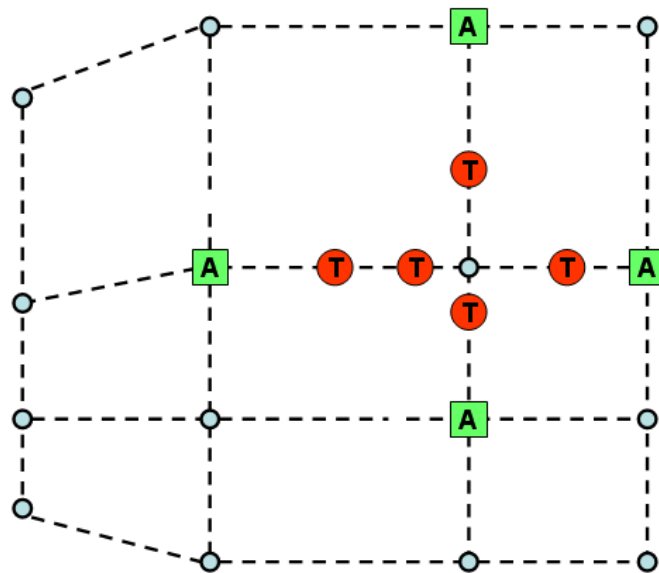


Figure 35: A case where all evasive targets are always observed.

CHAPTER VIII

CONCLUSION

This doctoral thesis studies the problem of how to place agents over a region of interest to track navigating targets. The problem arises in many real-life application domains such as urban search and rescue (USAR) in the aftermath of a disaster, security, and military surveillance and reconnaissance. The nature of the problem entails dynamic autonomous placement of agents to achieve an optimal or near-optimal coverage. Several approaches to the problem of interest have been developed in the past few decades. This thesis classifies existing approaches according to three independent criteria: the multiplicity of agents and targets, the centrality of the approach, and the predictiveness of the approach. The work presented in this thesis is shown to belong to the multi-agent, multi-target, distributed, and predictive classes. Therefore, according to each classification criterion, the work presented in this thesis falls into the hardest category, and thus can easily be adapted to the simpler ones.

The thesis introduces a general distributed hierarchical approach applicable to a team of autonomous agents that are capable of navigating across the region of interest and communicating among themselves. The approach entails that agents communicate at several levels of abstraction to fuse their sensor data, coordinate their decisions, and share their knowledge. Although this doctoral thesis uses surveillance applications in urban environments as a testbed, the approach is believed to be general and can be applied to a wide variety of applications.

The distributed approach is applied to two different types of urban terrain zones: low-rise, widely spaced and high-rise, closely spaced. For each type, the agent placement problem is formulated, a target motion model is defined, and a number of algorithms are developed. Since some of the algorithms assume a stochastic model that describes target motion (which

may not be available prior the surveillance mission), an novel online target motion estimation is developed. The distributed nature of the approach is stressed throughout the thesis. The algorithms are described in their distributed form, where an agent goes through rounds of local processing and communication with other agents. The algorithms are capable of tolerating isolated agent failures. Simulation results suggest that the developed approach is applicable to the two types of urban terrain zones providing near-optimal coverage in each case.

This doctoral thesis is believed to have contributed the following:

- a distributed hierarchical approach to agent placement applicable to a wide variety of target tracking applications,
- a study of how to apply the approach to two types of urban terrain zones,
- a non-model-based distributed heuristic algorithm (CEP) that outperforms the best-existing counterpart,
- a model-based algorithm (PCEP) that improves over the performance of CEP given a stochastic target motion model,
- a novel model estimation approach that can be applied in real time to estimate the target motion model of targets from sensor observations,
- a graph-theoretic reformulation for the agent placement problems for urban environments,
- a distributed algorithm for the agent placement problem in urban environments with a predefined performance guarantee,
- an optimal distributed algorithm that minimizes the average power consumed by agents for navigation purposes, and
- a practical dynamic programming algorithm for dealing with evasive targets.

REFERENCES

- [1] “Nist/sematech e-handbook of statistical methods.” Successfully downloaded from <http://www.itl.nist.gov/div898/handbook/> in Sep. 2004.
- [2] AGEEV, A. and SVIRIDENKO, M., “Approximation algorithms for maximum coverage and max cut with given sizes of parts,” in *Proc. of the Seventh International IPCO Conference on Integer Programming and Combinatorial Optimization*, (London, UK), pp. 17–30, Springer-Verlag, 1999.
- [3] ARKOV, V., KULIKOV, G., and BREIKIN, T., “Fuzzy Markov modeling in automatic control of complex dynamic systems,” in *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems*, (Trieste, Italy), pp. 287–289, 1999.
- [4] ASTROM, K., *Introduction to Stochastic Control Theory*. New York, NY, USA: Academic Press, 1970.
- [5] BARTAL, Y., BYERS, J., and RAZ, D., “Global optimization using local information with applications to flow control,” in *Proc. of the Thirty Eighth Annual Symposium on Foundations of Computer Science (FOCS '97)*, pp. 303–312, 1997.
- [6] BARTO, A., BRADTKE, S., and SINGH, S., “Learning to act using real-time dynamic programming,” *Artificial Intelligence*, vol. 72, pp. 81–138, 1995.
- [7] BATALIN, M. and SUKHATME, G., “Sensor coverage using mobile robots and stationary nodes maxim,” in *Proc. of the SPIE*, pp. 269–276, Aug. 2002.
- [8] BATALIN, M. and SUKHATME, G., “Spreading out: a local approach to multi-robot coverage,” in *Proceedings of the Sixth International Symposium on Distributed Autonomous Robotic Systems*, (Fukuoka, Japan), pp. 373–382, 2002.
- [9] BELLMAN, R., *Adaptive Control Processes*. Princeton, NJ, USA: Princeton University Press, 1961.
- [10] BERCHTOLD, A., “Estimation of the mixture transition distribution model.” Technical Report No. 360, Department of Statistics, University of Washington, Seattle, WA, 1999. Successfully downloaded from <http://www.stat.washington.edu/www/research/reports/1999/tr360.ps> in Sep. 2004.
- [11] BERCHTOLD, A. and RAFTERY, A., “The mixture transition distribution model for high-order Markov chains and non-Gaussian time series,” *Statistical Science*, vol. 17, no. 3, pp. 328–356, 2002.
- [12] BERTSEKAS, D., “The auction algorithm: A distributed relaxation method for the assignment problem,” *Annals of Operations Research*, vol. 14, pp. 105–123, 1988.

- [13] COHEN, I. and MEDIONI, G., “Detecting and tracking objects in video surveillance,” in *Proc. of the IEEE Computer Vision and Pattern Recognition*, (Fort Collins, CO, USA), June 1999.
- [14] COOK, D., GMYTRASIEWICZ, P., and HOLDER, L., “Decision-theoretic cooperative sensor planning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 10, pp. 1013–1023, 1996.
- [15] COOK, D., GMYTRASIEWICZ, P., and HOLDER, L., “Decision-theoretic multi-agent sensor planning,” *Reconnaissance, Surveillance, and Target Acquisition for the Unmanned Ground Vehicle*, pp. 413–428, 1997.
- [16] CULLER, D., SINGH, J., and GUPTA, A., *Parallel Computer Architecture : A Hardware/Software Approach*. San Francisco, CA, USA: Morgan Kaufmann, first ed., 1998.
- [17] ELLEFSEN, R., “Aerospace operations in urban environments: Exploring new concepts.” Report prepared for Naval Surface Warfare Center, Dahlgren Laboratory, Dahlgren, VA, 1999.
- [18] GAGE, D., “Command control for many-robot systems,” in *Proc. of the Nineteenth Annual AUUVS Technical Symposium*, (Huntsville, AL, USA), pp. 22–24, 1992.
- [19] HAN, Q., YE, Y., ZHANG, H., and ZHANG, J., “On approximation of max-vertex-cover,” *European Journal of Operational Research*, no. 2, pp. 207–220, 2002.
- [20] HOCHBAUM, D., *Approximate Algorithms for NP-Hard Problems*. PWS Publishing Company, 1995.
- [21] ISLER, V. and KANNAN, S., “Randomized pursuit-evasion with limited visibility,” in *Proc. of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, (New Orleans, LA, USA), 2004.
- [22] ISLER, V., KANNAN, S., and KHANNA, S., “Randomized pursuit-evasion with limited visibility,” in *Proc. of ACM-SIAM Symposium on Discrete Algorithms*, pp. 1053–1063, 2004.
- [23] KUHN, H., “The Hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.
- [24] LATOMBE, J., *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.
- [25] LAVALLE, S., GONZLEZ-BANOS, H., BECKER, C., and LATOMBE, J., “Motion strategies for maintaining visibility of a moving target,” in *Proc. of the IEEE International Conference on Robotics and Automation*, pp. 731–736, 1997.
- [26] LIPTON, A., FUJIYOSHI, H., and PATIL, R., “Moving target classification and tracking from real-time video,” in *Proc. of the Fourth IEEE Workshop on Application of Computer Vision*, (Princeton, NJ, USA), pp. 8–14, Oct. 1998.
- [27] NORRIS, J., *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge, UK: Cambridge University Press, 1998.

- [28] O’ROURKE, J., *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.
- [29] PARKER, L., “ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots,” in *Proc. of 1994 the IEEE/RSJ/GI International Conference on Intelligent Robotics Systems*, (Munich, Germany), pp. 776–783, Sep. 1994.
- [30] PARKER, L., “ALLIANCE: An architecture for fault-tolerant multi-robot cooperation,” *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.
- [31] PARKER, L., “Cooperative robotics for multi-target observation,” *Intelligent Automation and Soft Computing*, vol. 5, no. 1, pp. 5–19, 1999.
- [32] PARKER, L., “Multi-robot learning in a cooperative observation task,” in *Proc. of the Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS)*, pp. 391–401, 2000.
- [33] PARKER, L., “Distributed algorithms for multi-robot observation of multiple moving targets,” *Autonomous Robots*, vol. 12, no. 3, pp. 231–255, 2002.
- [34] PARKER, L., TOUZET, C., and JUNG, D., “Learning and adaptation in multi-robot teams,” in *Proc. of the Eighteenth Symposium on Energy Engineering Sciences*, pp. 177–185, 2000.
- [35] PUTERMAN, M., *Markov Decision Processes*. New York, NY, USA: Wiley-Interscience, 1994.
- [36] RAFTERY, A., “A model for high-order Markov chains,” *Journal of the Royal Statistical Society*, pp. 528–539, 1985.
- [37] TOUZET, C., “Robot awareness in cooperative mobile robot learning,” *Autonomous Robots*, vol. 8, no. 1, pp. 87–97, 2000.
- [38] VICK, A., STILLION, J., FRELINGER, D., KVITKY, J., LAMBETH, B., MARQUIS, J., and WAXMAN, M., *Aerospace Operations in Urban Environments: Exploring New Concepts*. Santa Monica, CA, USA: RAND, 2000.
- [39] VIDAL, R., SHAKERNIA, O., KIM, H., SHIM, D., and SASTRY, S., “Probabilistic pursuit evasion games: Theory, implementation, and experimental evaluation,” *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 662–669, Oct. 2002.
- [40] WERGER, B. and MATARIC, M., “Broadcast of local eligibility: Behavior-based control for strongly cooperative robot teams,” in *Proc. of the Fourth International Conference on Autonomous Agents*, pp. 21–22, 2000.
- [41] WERGER, B. and MATARIC, M., “From insect to internet: Situated control for networked robot teams,” *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 173–197, 2001.
- [42] WEST, D., *Introduction to Graph Theory*. Prentice Hall, second ed., 2000.
- [43] WINELD, A., “Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots,” *Distributed Autonomous Robotic Systems 4*, pp. 273–282, 2000.

- [44] YULE, G., “On the theory of correlation for any number of variables, treated by a new system of notation,” in *Proc. of Royal Society*, vol. 79 of *A*, pp. 182–193, 1907.

VITA

Tamir A. Hegazy was born in Egypt in 1975. He received his B.S. degree in electronics engineering with a major in computer and control engineering from Mansoura University, Egypt in 1997. He is the recipient of the “Honor Degree” and the “Excellence Prize” from Mansoura University for his distinguished academic performance during the B.S. program. He received his M.S. degree in computer engineering at the Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, USA in 2001 on the topic of proactive resource allocation for real-time distributed systems. He is currently a Ph.D. candidate at the department of Electrical and Computer Engineering, Georgia Institute of Technology, USA. He has several publications in IEEE journals and conferences on a number of topics, including agent placement for target tracking and resource allocation for real-time distributed systems.

Tamir A. Hegazy will shortly be awarded his Ph.D. degree and will join the faculty of Georgia Institute of Technology as a postdoctoral associate at the Center for Process Systems Engineering. Afterwards, he will be working as an assistant professor at the department of Computers and Systems Engineering, College of Engineering, Mansoura University, Egypt.