UAV SWARM COORDINATION AND CONTROL FOR ESTABLISHING WIRELESS CONNECTIVITY

ACHUDHAN SIVAKUMAR

NATIONAL UNIVERSITY OF SINGAPORE

2011

UAV SWARM COORDINATION AND CONTROL FOR ESTABLISHING WIRELESS CONNECTIVITY

ACHUDHAN SIVAKUMAR

Bachelor of Computing (Computer Engineering)

School of Computing, National University of Singapore

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2011

Abstract

This thesis addresses the vital problem of enabling communications in a disaster struck area. Emphasis is placed on the need for data communication between various points on the ground, which cannot be effectively established in a short time frame using existing methods. We propose the use of completely autonomous Unmanned Aerial Vehicles (UAVs) mounted with wireless equipment to accomplish this goal by coordinating themselves to build a wireless backbone for communication. The problem then becomes one of coverage, search and tethering, where a swarm of UAVs (agents) are required to cooperatively cover a given area and search for ground nodes while also relaying packets between already found ground nodes. In this thesis, we explore the above problem from two main perspectives - 1) A theoretical perspective that identifies what can be done with complete *a priori* information, and 2) A realistic, practical perspective that demands a decentralized solution under realistic networking and environmental conditions.

For the theoretical perspective, we take a geometric approach to design paths for agents with the aim of minimizing maximum latency in the network. We propose Bounded Edge-Count Diametric Latency Minimizing Steiner Tree (BECDLMST) as a solution structure capable of achieving very low maximum latency. The concept of BECDLMST is based on the concept of minimal Steiner trees in geometry, which are known to provide the shortest interconnect between any given set of nodes. BECDLMST builds on this idea to generate agent paths such that agent

travel distances are lowered, which in turn lower maximum network latency. We go on to show that finding the optimal BECDLMST is an NP-hard problem. So we first provide an exact exponential algorithm to find the best BECDLMST, and then devise an efficient approximation through an anytime heuristic. Although exponential in nature, the exact algorithm ensures that the solution space is pruned as much as possible at every step. The approximation on the other hand utilizes ideas from particle swarm optimization to generate a near optimal BECDLMST in quadratic time. As such, a Minimum Diameter Steiner Tree (MDST) is iteratively evolved to produce a network structure that minimizes the maximum latency. Experimental results on computation time and resulting network latency are presented for both algorithms. The contribution of the theoretical analysis is a solution structure that can be the target as well as the basis for comparison for other decentralized algorithms.

In looking at the problem from a practical perspective, we identify a number of challenges to be addressed, namely: 1) Lack of global information in online agent planning, 2) Intermittent and mobile ground nodes, 3) Opposing trade-offs in a dynamic environment, 4) Limited communication bandwidth, and 5) Adverse wind effects. To this end, we propose a suitable hierarchical, decentralized control and coordination architecture. A robust control algorithm is developed to ensure precise waypoint navigation of UAVs. This in turn is shown to lay the foundation for a multiagent coordination algorithm that can afford to not consider adverse wind effects within operational limits. A communication-realistic, dynamically adaptive, completely decentralized, agent-count-and-node-count-independent coordination algorithm is presented that has been empirically shown to non-monotonically increase a performance metric, Q, through time. The performance metric, Q, takes into consideration, the average cell visit frequency, average node service time, and packet latency to determine the performance of the system. The approach taken is

"near-decision-theoretic", in the sense that each agent tries to maximize a scoring function, without a fixed horizon and with the lack of stochastic models to describe the environment. The decision algorithm for relaying packets is designed so that agent paths mimic certain characteristics of BECDLMST. Simulations show that the decentralized control and coordination algorithm achieves very promising latency results that are inferior to the centralized version by only 10-50%. Experimental results illustrating the adaptive behavior of the agents and the resulting performance in terms of network latency and search quality are presented.

Given that one of the main aims of this thesis is to develop a solution that can be practically deployed, we perform field tests to prove the performance of our autonomous control system as well as the viability of air-to-ground and air-to-air communication, which forms the very basis for our proposed solution. Apart from numerous successful flight tests, hardware-in-the-loop simulations are also conducted to evaluate performance in a controlled manner.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Colin Tan. I consider it a blessing to have got the opportunity to work with Colin for over 5 years starting right from my final year project all the way through my Ph.D. Starting from Embedded Systems in my undergraduate second year, Colin has taught me an incredible lot, not only academically, but also about life. His guidance, encouragement and support are what have made this thesis possible. I will never in my life forget his advice and help. For being a great mentor, an understanding supervisor, an encouraging friend, and a motivating role model, I'll forever be grateful to Colin.

I would like to thank Dr. Winston Seah, who provided me guidance through the beginning stages of my research. His initial project is what led me towards the research focus of this thesis. My gratitude also goes to Dr. Bryan Low for all the discussions and exchange of ideas.

I would also like to thank everybody who has contributed to parts of the project in some way - Phang Tze Seng, for his work extending and validating my control algorithms; Winson Lim, for his contributions and help towards getting the UAVs in the air; Eddie Tan, Eric Toh, and Teo Keng Boon, for their contribution towards the networking component during flight tests; and Kalvin Lim, for his invaluable piloting skills.

I will always be grateful to my mother and brother, who have been the incredible pillars of support and unlimited source of encouragement all through my studies and beyond. Without their contribution in my life, I could never imagine seeing myself where I am.

I am also very thankful to my two great seniors - Ramkumar Jayaseelan and Unmesh Bordoloi - who guided me at various stages of my research. Their inspiration helped me cross a number of barriers.

My long years in NUS would have been impossible to get by without my good friends - Jesse Prabawa, Alex Ngan, Arik Chen, Bennette Teoh, Brandon Ooi, Deepak Adhikari, Dulcia Ong, Edwin Tan, Fong Hong, Huajing Wang, Huiyu Low, Jingying Yeo, Sharad Arora, and Tai Kai Chong - who have always been there when I needed them.

Finally, I would like to thank Mdm Loo Line Fong, Mark Bartholomeusz and all the admin staff from the School of Computing, who have been of great support through my last 4 years in NUS.

Contents

List of Publications

1. A. Sivakumar, T. S. Phang, and C. K. Y. Tan. Stability Augmentation for Crosswind Landings using Dynamic Cell Structures. *In Proc. AIAA Guidance, Navigation and Control Conf.*, (AIAA GNC'08), Paper 2008-6467, Honolulu, Hawaii, Aug 2008.

2. A. Sivakumar, T. S. Phang, C. K. Y. Tan, and W. K. G. Seah. Robust Airborne Wireless Backbone using Low-Cost UAVs and Commodity WiFi Technology. *In Proc. IEEE Intelligent Transport System Telecommunications Conf.*, (ITST'08) Phuket, Oct 2008.

3. A. Sivakumar and C. K. Y. Tan. Formation Control for Lightweight UAVs Under Realistic Communications and Wind Conditions. *In Proc. AIAA Guidance, Navigation and Control Conf.*, (AIAA GNC'09), Paper 2009-5885, Chicago, Aug 2009.

4. A. Sivakumar and C. K. Y. Tan. UAV Swarm Coordination using Cooperative Control for establishing a Wireless Communications Backbone. *In Proc. 9th Intl. Conf. Autonomous Agents and Multiagent Systems*, (AAMAS'10), Toronto, May 2010.

5. A. Sivakumar and C. K. Y. Tan. Anytime heuristic for determining agent paths that minimize maximum latency in a sparse DTN. Short. *To appear*

in Proc. 23rd IEEE Intl. Conf. on Tools with Artificial Intelligence, (ICTAI'11), Florida, Nov 2011.

6. A. Sivakumar and C. K. Y. Tan. Circular trajectory tracking by lightweight UAVs in the presence of winds. *To appear in Proc. 7th IEEE Intl. Conf. on Intelligent Unmanned Systems*, (ICIUS'11), Chiba, Japan, Nov 2011.

List of Figures

List of Tables

List of Abbreviations

BDBCST	–	Bounded Diameter Bounded Cost Spanning Tree
BECDLMST	–	Bounded Edge Count Diametric Latency Minimizing Steiner Tree
bmu	–	best matching unit
COTS	–	Commercial Off-the-shelf
DARD	–	Density Aware Route Design
DCS	–	Dynamic Cell Structure
Dec-POMDP	–	Decentralized Multiagent Partially Observable Markov Decision Process
DTN	–	Delay Tolerant Network
FRA	–	Ferry Relay Algorithm
FSM	–	Finite State Machine
GA	–	Genetic Algorithm
GPS	–	Global Positioning System
HB	–	Hybrid state
HWIL	–	Hardware in the loop
IFCS	–	Intelligent Flight Control System
LRP	–	Linking Rendezvous Point
l-SCFR	–	logarithmic-Store Carry Forward Routing
MANET	–	Mobile Ad Hoc Network
MDST	–	Minimum Diameter Steiner Tree
MEC	–	Minimum Enclosing Circle

MRP – Merging Rendezvous Point

MRT – Multiple Routes Topology

NDI – Nonlinear Dynamic Inversion

NRA – Node Relay Algorithm

PID – Proportional Integral Derivative

PR – Proxy state

PSO – Particle Swarm Optimization

PTPM – Perfectly Topology Preserving Map

RL – Relay state

RP – Rendezvous Point

SA – Simulated Annealing

SR – Search state

SRT – Single Route Solution

TSP – Traveling Salesman Problem

UAV – Unmanned Aerial Vehicle

WNCS – Weighted Node Cover Subset

Chapter 1

Introduction

1.1 Motivation

The response phase in disaster management plays a key role in mitigating possible adverse effects including loss of lives. Part of the response phase involves the dispatch of rescue teams (on ground) into the disaster area to survey the damage and find survivors. These rescue teams often need to send data back to the base station or to other rescue teams in the area, including information like images, videos, or even calls for additional support. Moreover, communication between the rescue teams and with the base station can greatly enhance coordination between the various teams. Unfortunately, in a disaster situation, normal communication infrastructure tends to be damaged or destroyed. Traditionally, push-to-talk services [1] have been used for voice communication between base stations and rescue teams. However, such services are not designed to handle data communications involving images, videos, sensor readings, etc., that require higher bandwidths in the order of a few Mbits per second. Attempts have been made to use satellite communications for exchange of information between first responders [2]. However, satellite communications through services like Iridium [3] provide very low band-

widths in the range of 10kbps. This scarce bandwidth would have to be shared by multiple rescue teams in the same area, thus making the available bandwidth for each team, extremely small. The problem then requires a solution that:

1. Can establish communications with minimal setup time
2. Costs little and can be built easily
3. Handles the bandwidth requirements of data communication

We believe that WiFi-mounted Unmanned Aerial Vehicles (UAVs) have the potential to provide a feasible solution to the above problem. The challenge as to how this can be done is what motivates this thesis. We suggest and work with WiFi as opposed to other communication modes owing to their ease of availability off-the-shelf and common presence in numerous devices. Theoretically WiFi could be replaced with other wireless means of communication such as 3G and GPRS as well.

1.2 Delay Tolerant Networking and UAVs

Disaster struck regions tend to span huge areas with survivors and rescuers dispersed in a sparse manner. Now building a fully connected wireless network over the entire area would need immense resources and would not be practically feasible. Numerous routing algorithms such as Dynamic Source Routing [4] and Location Aided Routing [5] have been developed for data delivery in wireless ad hoc networks. The current algorithms make the assumption that the network graph is connected and fail to route messages if there is not a complete route from source

to destination at the time of sending. Under these conditions, most existing routing algorithms will fail to deliver messages to their destinations since no route is found due to network partition.

As a result, we turn to Delay Tolerant Networking (DTN) [6], which is a relatively new paradigm of networking that deals with scenarios involving the lack of continuous connectivity between packet sources and packet destinations. DTNs were originally introduced as a solution to communication in space. However, many of the ideas have been directly applied to earthbound networks that exhibit a lack of continuous connectivity. DTNs have been an area of intense interest in the networking community with numerous works proposing and studying routing mechanisms at all the various networking layers for different network mobility models ([7] provides a detailed survey). DTN routing methods are referred to as mobility-assisted routing [8] that employs the *Store-Carry-Forward* model. The basic requirement for the viability of DTNs is mobility and usually at least one of the following two cases of mobility is assumed:

1. Nodes in the network are mobile and come in contact with other nodes from time to time

2. Special mobile agents physically carry-store-relay-and-deliver packets

In our problem context, there is a need to proactively build a communications backbone and not depend on the movement of ground nodes. As a result, the second of the above two categories of mobility would be the appropriate match.

The special mobile agents in this case need to be able to maneuver in a disaster struck area and at preferably high speeds. We believe Unmanned Aerial Vehicles (UAVs) would be ideal candidates to act as mobile agents as they are airborne and agile. The envisioned solution would then be to deploy a set of UAVs, each

mounted with a wireless communication device like a WiFi antenna, so as to build a wireless backbone over which various entities on the ground such as rescue teams, relief agencies, survivors, first responders, etc. can communicate. The use of UAVs for this purpose is further motivated by recent works that have shown UAVs to be effective for complex tasks such as diffuse gas and plume detection [9, 10], coordinated search and reconnaissance [11, 12], *in situ* atmospheric sensing [13, 14], and as agents in the battlefield [15, 16]. These works have highlighted the advantages of using cooperative teams of autonomous UAVs, namely:

- parallel functioning to accomplish a task in shorter time and provide greater sensor coverage
- robustness and fault-tolerance even in cases of vehicle loss
- low cost of groups of small UAVs as compared to larger aircrafts or satellites

We believe the same advantages would apply to the task of establishing communication between multiple ground nodes. In fact, the use of UAVs as communication relays is further justified and thus motivated by works that have used real experiments to prove the viability of air-to-ground communication through commercial off-the-shelf (COTS) 802.11 equipment (demonstrated in [17] as well as our field tests detailed in Chapter 10).

In the envisioned solution, a system of UAVs would provide a mobile ad hoc network (MANET) connecting ground devices like laptops, PDAs, cell phones, and any other communication device capable of wireless communication. It would be desirable to have the UAVs function autonomously and coordinate among themselves to establish one such communications network that can support high bandwidth data communications. In the event of a disaster, it should be possible to deploy a number of UAVs into the area and restore WiFi connectivity to both

survivors and rescuers within minutes. Although the motivating application is the enabling of communications between multiple ground nodes over a disaster struck area, we believe the solution can be directly applied to other scenarios like data relay for sparse wireless sensor networks and battlefield communications.

1.3 Research objective

The broad objective of this research is to study and explore the problem of how to move the UAVs so as to build a network. The specific questions we address are as follows:

1. What is the best course of action if the locations of ground nodes were known *a priori*?
2. If the locations were unknown, how can the combined problem of search and tethering (i.e. enabling ground node access to the network) be addressed?
3. How to coordinate among the UAVs in a decentralized manner with communication limitations for practical deployment?
4. How to achieve accurate navigation and control of UAVs to execute movement decisions?

The aim of this thesis then is to study and present policies and algorithms for the UAVs to autonomously control and coordinate themselves, so as to establish an efficient wireless communications backbone. The metric in particular that we are concerned with, is the maximum pairwise latency in the network. By minimizing this quantity, we aim to lower the upper bound on network latency. The motivation for dealing with maximum latency as opposed to average latency, is that a number

of networking applications such as streaming, can benefit from knowing the exact upper bound on latency. In the case of multimedia streaming for example, the upper bound on latency can be used to determine how long to buffer before ensuring a smooth playback at the receiver. If the upper bound on latency is lowered, the buffer time is also effectively lowered.

1.4 Overview of the Thesis

In this thesis, we try to explore the problem from different perspectives. In particular two perspectives are considered - 1) A theoretical perspective that explores centralized solutions with complete *a priori* information, and 2) A practical perspective that explores decentralized solutions with no *a priori* information and under realistic constraints. As far as possible, the aim is to provide a complete study of the problem and its applicable solutions. In our theoretical analysis, we propose a solution structure to minimize maximum network latency as well as two algorithms (exact and approximate) to derive it. Subsequently, we discuss how the problem expands into one with a lot more challenges when considered in a realistic scenario. We then go on to address these challenges and design a complete control and coordination system that is deployment-ready. We conclude the thesis with results on real flight tests that were performed to validate the system.

1.5 Thesis Contributions

1. **Solution structure giving agent paths for minimizing maximum latency in a network of sparsely located stationary ground nodes.** We approach the problem of establishing a network from the theoretical perspective and derive a centralized solution had the ground node locations

been known *a priori* (i.e. with perfect information). The novel proposed solution structure, termed Bounded-Edge Count Diametric Latency Minimizing Steiner Tree (BECDLMST) is shown to achieve network latencies far lower than existing methods. An exact exponential algorithm is presented to find such a solution for any given ground node setting.

2. **Efficient anytime heuristic to determine the BECDLMST for any given set of nodes and agents.** An efficient approximating algorithm is presented as a practical alternative to the exponential algorithm mentioned in the first contribution. As a result, this thesis contributes an efficient centralized method (that is also anytime in nature) to generate latency minimizing agent paths when ground node locations are known.

3. **An efficient solution for the novel problem of coverage, search and tethering, combined, under realistic wind conditions and communication limitations.** We provide a control and coordination solution that balances the tasks of search and relay, while minimizing latency and maximizing visit frequency. Importantly, this is achieved in a realistic setting with decentralized control, without global information at each agent, regardless of intermittency of ground stations, in the presence of winds, and under realistic communication limitations.

4. **A reactive control system capable of achieving accurate waypoint navigation despite adverse crosswind effects.** The control component presented in the thesis introduces a novel system that works reactively using the normally-unused cross-track parameter along with a neural network, as opposed to existing solutions that require hard-to-obtain measurements of wind speed and direction. It allows for realistic implementation of other higher level coordination algorithms that use waypoint navigation but do not consider wind effects.

5. **A method to realize the idea of using UAVs to build a wireless backbone for multiple ground stations.** This is a practical contribution of this thesis. The control and coordination algorithms are deployment-ready owing to the consideration of realistic conditions. The control algorithms as well as air-air and air-ground communication are field tested and proven to be viable. Further testing might be required for swarm-scale UAVs, but no restrictions on this approach have be discovered.

6. **A bandwidth-minimizing belief exchange mechanism for UAV-based multiagent coordination.** The belief exchange mechanism proposed as part of our coordination algorithm can be applied to many other applications using UAV swarms. The novel idea of using the limitations of UAV motion in choosing grid cells that encompass enough information to interpolate missing cell information is applicable to situations other than the specific problem considered in this thesis.

1.6 Thesis Outline

The chapters of this thesis are organized as follows:

Chapter 2 presents a formal definition of the agent path design problem for minimizing network latency without consideration for practical aspects such as wind or network imperfections. It is essentially a formal translation of the question of what can be done if complete information was available *a priori*. The problem is presented more formally as one requiring an algorithmic solution.

Chapter 3 discusses solutions to the problem described in Chapter 2. Existing literature comprising works related to this problem are first discussed. Subsequently,

our proposed solution structure, termed the Bounded Edge Count Diametric Latency Minimizing Steiner Tree (BECDLMST), is introduced. The BECDLMST is described in detail and certain advantages of its characteristics are discussed.

Chapter 4 then introduces an exact exponential algorithm to find the optimal BECDLMST. The algorithm for converting the problem at hand to one of Weighted Set Cover is discussed in detail.

Chapter 5 presents the algorithm proposed to overcome the exponential time complexity of the exact algorithm discussed in chapter 4. It deals with approximating the BECDLMST using techniques that run in polynomial time. An efficient anytime heuristic that utilizes ideas from Particle Swarm Optimization is presented to find near-optimal BECDLMSTs in quadratic time.

Chapter 6 proceeds with expanding the problem originally described in Chapter 2. Following the discussion of what can be done under perfect information, this chapter presents the challenges introduced by considering the same problem under realistic conditions. The problem first described in Chapter 2 is now modified and expanded to incorporate the additional challenges.

Chapter 7 discusses the solution overview and compares it against other work in this area. A hierarchical control and coordination architecture is proposed and presented here.

Chapter 8 delves into the control aspect of the problem and discusses the part of the solution that enables precise navigation. A method using neural networks (specifically, Dynamic Cell Structures) to correct for wind effects is proposed and described in depth.

Chapter 9 then details the proposed multiagent coordination scheme for enabling

the wireless backbone in the context of the problem described in Chapter 6. A decentralized realistic solution using a near-decision-theoretic approach is discussed.

Chapter 10 details the real life experiments that were conducted to validate the control component and UAV-UAV and UAV-ground communications. Methods used in deploying our algorithms on the aircraft are discussed and results from field tests are presented.

Chapter 11 concludes the thesis and presents possible directions for future work.

Chapter 2

Problem Definition

The overall objective of this thesis as laid out in Section 1.3 is addressed in phases. We first consider the problem from a theoretical standpoint to explore what can be done if ground node locations were known *a priori* and node intermittency and movement and network imperfections and control challenges by winds were absent.

Nomenclature:

M	number of agents
N	number of ground nodes
g_i	position of i^{th} ground node $\forall i = 1, 2, \ldots N$
n_h	number of hops (edges) on maximum latency path
λ_h	maximum hop length in the entire network
MEC	Minimum Enclosing Circle around all ground nodes
r_M	radius of MEC
c_M	center of MEC
v_{max}	maximum speed of any given agent

We consider a network of N sparsely-located, stationary ground nodes where packets can be sent from any node to any other node. Ground nodes are represented

as points on a 2-D plane. M agents are available that can freely move on the 2-D plane and physically carry and deliver packets. Agent-agent, node-agent, and agent-node packet transfers are allowed when they meet. Points where agents meet and exchange packets are referred to as rendezvous points (RPs). Latency of a packet is defined as the duration between its generation and its delivery, and is given by the sum of wait time at source, t_w, and transit time on the path from source to destination, t_p. We then define,

Problem1 : To find movement policies for each agent m, such that the maximum latency for any packet in the network is minimized.

We assume a sparse DTN wherein inter-node distances are large, thus making wireless communication ranges negligible. Packet transmission times are also considered to be negligible in comparison to transit time along path from source to destination. Although each ground node is represented as a point on a 2-D plane, it could refer to a gateway node in a fully-connected cluster of close nodes. We also assume a homogeneous set of agents capable of speeds up to v_{max}. For simplicity, we shall use the term network latency to refer to maximum latency in the network, for the rest of this thesis.

2.0.1 Network Traffic Model

For network traffic, we consider the worst case where every ground node is equally interested in communicating with every other ground node. Packets can be sent from any node to any other node at any time with equal probability. As a result, none of the node-node pairs can be ignored in the solution. Worst case maximum latency in the network is the maximum latency if every node was continuously sending packets to every other node. Such a network traffic model is referred to as the uniform traffic model [18].

Chapter 3

Solution characterization under perfect information

In this chapter we analyze the characteristics of a solution to the agent path design problem described in Chapter 2. We first take a look at approaches that have been presented in related literature. Subsequently, we approach the problem from a geometric perspective and describe a solution structure that can give us the required agent paths. The characteristics identified in this chapter are later used in devising algorithms (exact and approximate) in Chapters 4 and 5.

3.1 Motivation

There are two reasons for analyzing the problem under perfect information. It presents the best case scenario in terms of uncertainty and studies the network performance that can be achieved under such circumstances. The results then provide a target and a scale for comparison for our online solution (as well as others' solutions) that work with incomplete information.

The second reason is that if a solution structure can be determined for the problem under perfect information, applicable characteristics from this solution can be applied to the problem under imperfect information. In other words, we believe that the offline solution with *a priori* information can provide insight into designing an online solution.

3.2 Related work

In this section, we shall explore works in the area of offline agent path planning under perfect information. The case of online solutions under changing and imperfect information is discussed in Chapter 9, with related works explored in Section 9.1. Offline solutions are often characterized based on how packets are relayed on their paths from source to destination. In particular, three such categories have been identified:

1. No relay - Packets travel from source to destination on the same agent without any transfers

2. Node relay - Agents interact with each other indirectly through ground nodes. Ground nodes buffer and relay packets between ferries

3. Agent relay - Agents interact with each other directly by transferring packets between each other when they rendezvous

The works under each of the categories are discussed in more detail below.

3.2.1 No relay

Solutions that use no relaying mechanism originated in the context of single agent situations. A number of past works have dealt with the use of a single agent (also known as message ferry [19, 20, 21, 22, 23] or mobile element [24, 25, 26, 27, 28]) to forward messages in DTNs. We know that given a number of sparsely located ground nodes and a single agent, the fastest way to traverse through all the nodes, is given by the corresponding Travelling Salesman Problem (TSP) tour. However a number of these works consider scenarios where ground nodes are located close enough that wireless communication range cannot be ignored. The problem then gets transformed to that of clustering and connecting the clusters using the single ferry [29]. On the other hand, a number of works in this category deal solely with wireless sensor networks where a single mobile element collects data from all sensors and delivers it to a single sink node. The nature of this problem varies from our problem where all pairwise communications are possible.

When extending the single agent solution to incorporate multiple agents, one of the approaches used is to retain the same path design and add more agents along the single agent path [30, 23]. These approaches do not utilize the advantages of relaying and distribution of load. In general, such solutions are categorized as single route (SRT) solutions. The main disadvantage of SRT solutions is the non-utilization of possible interactions between agents. They often lead to extremely high packet latencies in sparse DTNs.

3.2.2 Node relay

The usage of ground nodes to buffer and relay packets is an approach that a number of works use to allay the complexities of synchronization. When a ground

node buffers a packet, it can hold on to it until the appropriate agent picks it up. The complexity of having to choose rendezvous points for agents to meet is eliminated. Essentially, ground node locations are treated as rendezvous points that do not need synchronized meetings between agents. The specific approaches under this category differ in their assumptions about the ground nodes and their locations. The three most prominent works that use the node relay approach are Node-Relay-Algorithm (NRA) [23], Multiple-Routes-Topology (MRT) [30], and Density-Aware-Route-Design (DARD) [31].

NRA first divides the entire area into a grid of an arbitrary number of rows and columns such that at least one agent is available for each occupied grid cell. The nearest nodes to each edge in the grid cells act as the relay point between the two cells. MRT's approach to division of the area is slightly different from that of NRA. They start dividing the area using vertical and horizontal slices such that each slice divides a given cell in a balanced manner w.r.t. number of nodes in the cell. The process is repeated until there's only agent available per cell. Choosing of relay nodes uses one of 2 options: MRT-Tree or MRT-Grid. MRT-Tree ensures that only one path exists from any given cell to any other cell. MRT-Grid goes along the lines of NRA to provide a relay node every 2 adjacent cells. Finally, DARD divides the nodes into clusters using the k-means algorithm and assigns one agent to each cluster. Considering each cluster as a individual vertex, a minimum spanning tree is built connecting the clusters. Gateway nodes are chosen to connect 2 adjacent clusters. Figure 3.1 illustrates what agent paths look like as generated by NRA, MRT and DARD respectively.

In these methods that use nodes as relays, the ground nodes are expected to be aware of the delay tolerant nature of the network and actively participate in buffering packets. In our motivating scenario, ground nodes can potentially be survivors with devices designed for regular wireless communication protocols. The

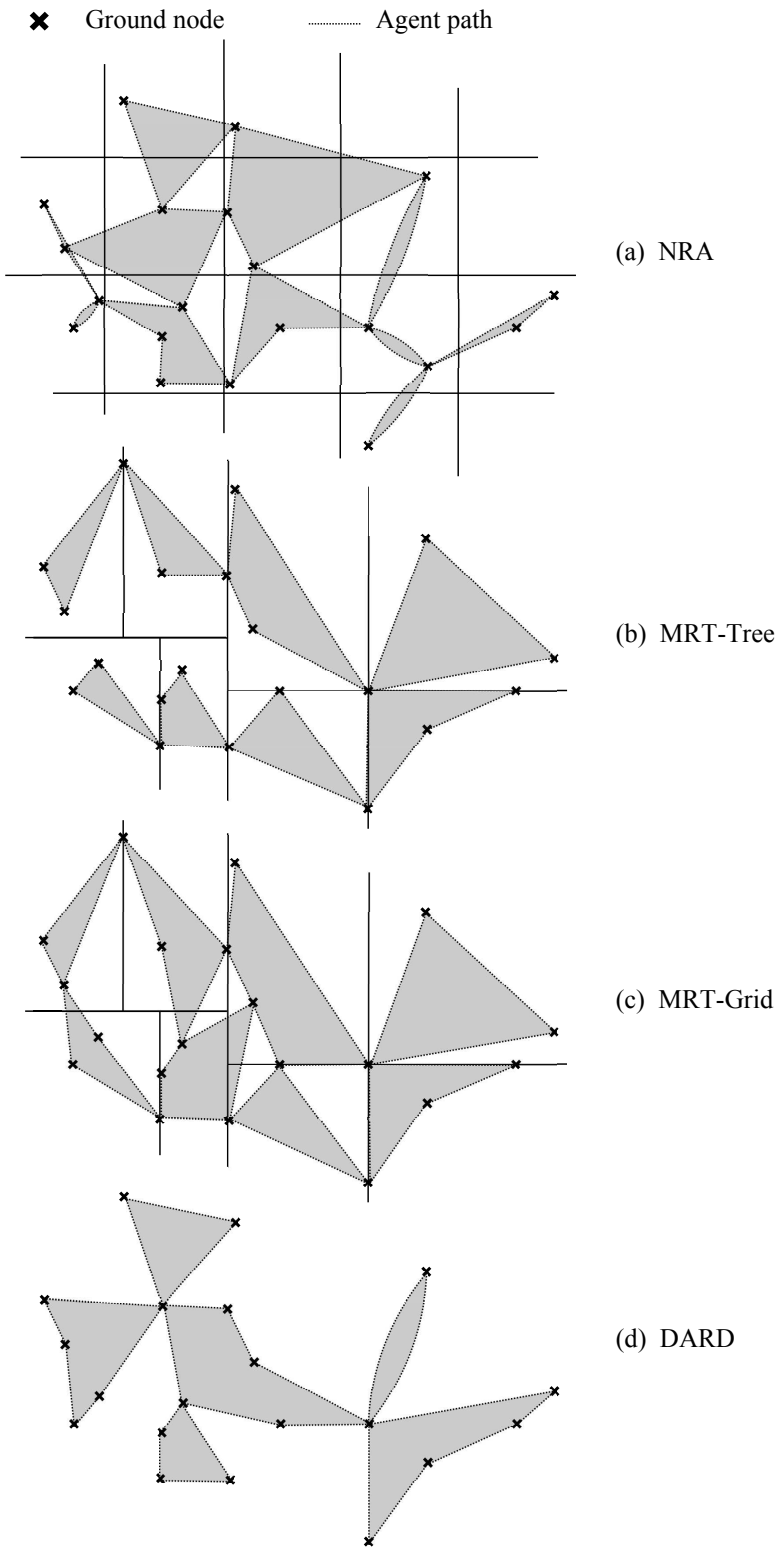


Figure 3.1: Agent paths as generated by (a)NRA[23], (b)MRT-Tree[30], (c)MRT-Grid[30], and (d)DARD[31] for a sample ground node configuration. The perimeter of each shaded region represents the path of an agent. The ground node at the intersection of 2 adjacent shaded areas acts as the relay between the 2 agents.

other main problem that affects the above methods is that although the burden of synchronization between agents is reduced, huge packet delays are introduced at each node. Nodes might not be the most strategic locations to exchange packets between agents. The time spent by a packet waiting at an intermediate node could have been spent moving closer to the destination to a more strategically chosen rendezvous point. This is where the agent relay approach comes in.

3.2.3 Agent relay

The agent relay approach refers to methods where agents directly exchange packets with another agent when they meet. Typically points where agents come in contact with each other are referred to as rendezvous points. Since both agents need to be present at a specified rendezvous point at the same time, the agents need to synchronize their visits to said rendezvous point. In order for one agent to not adversely affect another agent, rendezvous points must be intelligently determined. The two most prominent works in this area are logarithmic-Store Carry Forward Routing (l-SCFR) [32], and Ferry Relay Algorithm (FRA) [23].

l-SCFR treats the 2D space as a $k \times k$ mesh, where k is a arbitrary power of 2 dependent on the problem instance. Agents travel along the edges of the mesh and only intersection points on the mesh can act as rendezvous points. Agents adopt one of two roles - keeper or ferry. The keeper role is introduced to have stationary agents that hold packets at the rendezvous point until another agent picks it up. Agents switch between the two roles dynamically.

FRA, like NRA, divides the entire area into a grid. Subsequently, the midpoint of every cell edge is chosen as a rendezvous point. Each cell is serviced by one agent that passes through all nodes within that cell as well as the midpoints of its cell

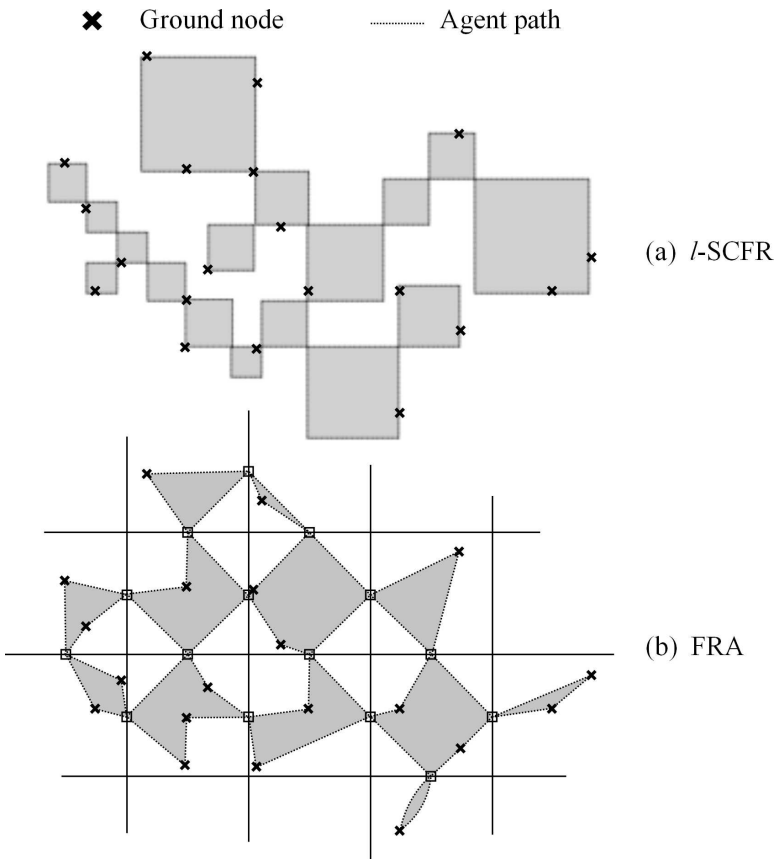


Figure 3.2: Agent paths as generated by (a)l-SCFR[32], (b)FRA[23] for a sample ground node configuration. The perimeter of each shaded region represents the path of an agent. The points where shaded areas meet are rendezvous points.

edges. The time taken between two consecutive rendezvous points is the same for every agent for the sake of synchronization. This also means that the bottleneck is the longest path between any 2 consecutive rendezvous points in any of the cells. Figure 3.2 shows examples of agent paths as generated by l-SCFR and FRA for a random ground node setting.

l-SCFR leads to either huge delays or the use of excessive agents owing to the need for keeper agents and the restricted movement along mesh edges. Owing to this restriction, packet latencies in l-SCFR tend to be higher than in FRA [23]. FRA suffers from unequal path lengths assigned for agents. The agent with the longest path length between consecutive rendezvous points acts as the bottleneck and slows all other agents down. In general, the main disadvantage of both these

methods is that rendezvous points are chosen arbitrarily.

3.2.4 Summary of related work

We saw existing methods that fall into one of the following three categories: No-relay, Node-relay, and Agent-relay. The discussion above showed us that of the three categories, Agent-relay based methods achieve lower network latencies than the other two categories. The reason is that in methods that do not use relaying, packets are never forwarded and thus every agent has to visit every single ground node, which leads to long paths. In methods that use node-relay, the node essentially acts as the rendezvous point for agents. Since node positions are fixed, the quality of rendezvous points is also fixed. Packets have high wait times at these ground nodes waiting for the next agent to pick them up. Agent-relay based methods overcome these problems by allowing packets to be forwarded from one agent to another. This way, packets are continuously moving from the time they are picked up at source, all the way until they are dropped off at destination. Of the two existing Agent-relay based methods, we saw how FRA provides networks with lower latencies as compared to l-SCFR owing to rectilinear agent movement and forced stationary agents in l-SCFR. Of all the methods studied, we conclude that FRA currently provides the best performance. The offline methods we propose in this thesis (Chapters 3-5) are therefore compared against FRA.

With the current agent-relay based methods, the problem we observed is that rendezvous points are chosen arbitrarily. In the first part of this thesis, we have taken up the task of determining strategic locations for such rendezvous points that can minimize maximum latency. We first present our proposed solution structure for the problem at hand. Subsequently, we provide an exact exponential algorithm as well as a heuristic based on ideas from Particle Swarm Optimization to closely

approximate the proposed solution structure.

3.3 Proposed solution structure

This section presents characteristics of our proposed solution structure that provides a set of agent paths that minimize maximum latency. We note that when inter-node distances are large, the main contributing factor to latency is the distance travelled by agents. The longer the distance, the more the time spent by a packet on the agent as well as time spent by a packet waiting for said agent. Briefly stated, the aim is then to reduce the distances traveled by agents.

Our proposed solution derives inspiration from the concept of Euclidean Steiner trees in geometry. For ease of reading, we shall refer to Euclidean Steiner tree as just Steiner tree for the rest of this thesis. Given a set, $\mathbb{G}$, of N points on a 2D plane, we know that the Steiner tree is a spanning tree with a set of vertices, $\mathbb{V} = \mathbb{G} \cup \mathbb{S}$, where $\mathbb{S}$ is any number of additional intermediate vertices known as Steiner points. Figure 3.3 shows an example of a Steiner tree. A minimal Steiner tree is one where the total cost of all edges (cost = Euclidean edge length) is minimized. It has been proven that a minimal Steiner tree provides the shortest possible interconnect between any set of points [33].

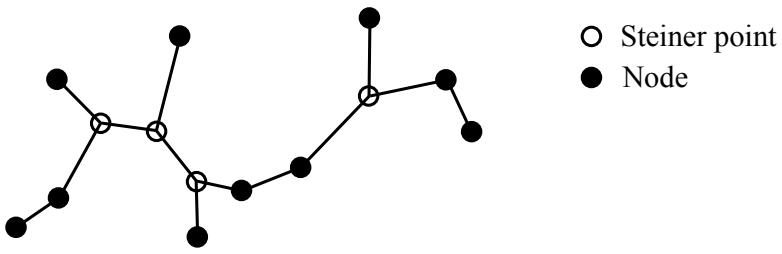


Figure 3.3: Example of a Steiner tree

We realize that in the problem at hand (described in Chapter 2), agents are required to span all N ground nodes, while traveling as little as possible. They are free to

rendezvous with other agents and exchange packets. These requirements have similar characteristics to the problem solved by the minimal Steiner tree. If the edges on the Steiner tree represented agent paths and the Steiner points represented rendezvous points, the minimal Steiner tree would represent the solution where the sum of distances traveled by all agents is minimized, while all ground nodes are serviced. However, the problem at hand has its differences, because rendezvous of agents needs to be synchronized and because the number of agents is M, which can be higher or lower than the number of edges in the minimal Steiner tree. Using these observations we propose the Bounded-Edge Count Diametric Latency Minimizing Steiner Tree (BECDLMST) as the solution to the problem described in Chapter 2. The next section describes in detail what the BECDLMST is.

3.3.1 BECDLMST

Description of BECDLMST

Our proposed solution is a Steiner tree that spans across all N ground nodes, with additional Steiner points representing rendezvous points. We know that the number of available agents is fixed at M and that each agent can potentially service more than one ground node. We therefore expand the traditional notion of an edge to "a path with one end at an intermediate vertex and another end at either an intermediate vertex or a ground node, that passes through zero or more ground nodes without any branches in between". With this notion of an edge, we say that every agent is assigned one edge, which is the path that it traverses back and forth, repeatedly and indefinitely. Every time a packet is picked up by an agent and dropped off to another agent or destination ground node, the packet is said to have traversed one hop. This definition of hop coincides with the notion of an edge (which is also an agent path), and hence these terms are used interchangeably. As

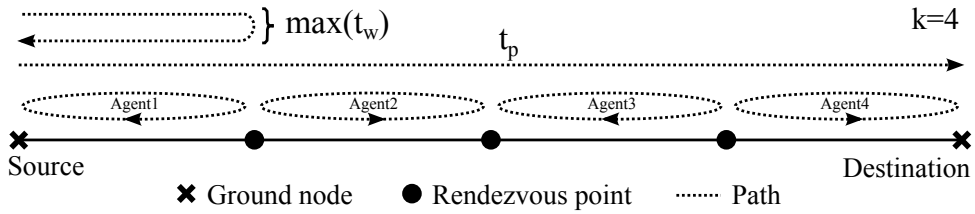


Figure 3.4: Maximum latency along a path [illustration uses k=4]

a result, what we are looking for is the Steiner tree that satisfies the limit on edge count (given by M) and minimizes maximum latency in the network.

Latency of a packet is defined as the duration between its generation and its delivery, and is given by the sum of wait time at source, t_w, and transit time on the path from source to destination, t_p. t_p depends on the number of hops the packet needs to travel and time spent on each hop. Referring to Figure 3.4, let a packet travel from Source to Destination using 4 hops. The maximum wait time ($\max(t_w)$), is given by the time taken by Agent 1 to complete one full back and forth cycle, which would have occurred if the packet was generated right after the agent left Source. t_p would be given by the sum of time taken along the 4 hops. Whenever an agent reaches a rendezvous point, it has to wait until the other agent(s) also reach the same rendezvous point so that packets can be transferred. In other words, rendezvous point visits must be synchronized between every two adjacent agents. As a result, the frequency at which each agent traverses its assigned path must be the same as the frequency at which its neighboring agents traverse their assigned paths. Since the Steiner tree is fully connected, that would mean that the traversal frequency of every agent in the entire tree has to be the same. Since all agents are assumed to be homogeneous with a maximum speed of v_{max}, the highest possible traversal frequency is limited by the bottleneck edge and given by, $\frac{v_{max}}{\lambda_h}$, where λ_h is the length of the longest edge/hop in the entire tree. Maximum latency (also called network latency), τ, can then be derived as follows:

Define,

t_w : Packet wait time at source

t_p : Packet propagation time along path from source to destination

n_h : number of edges/hops along the diameter of the tree

λ_h : length of the longest edge/hop in tree

v_{max} : maximum agent speed

τ : Maximum latency (network latency)

Then,

$$\begin{aligned} \max(t_w) &= \tfrac{2\lambda_h}{v_{max}} \\ \max(t_p) &= \tfrac{n_h\lambda_h}{v_{max}} \\ \tau &= \max(t_w) + \max(t_p) \end{aligned}$$

So we have,

$$\tau = \frac{1}{v_{max}}(n_h\lambda_h + 2\lambda_h) \tag{3.1}$$

where n_h is the number of edges/hops along the diameter of the tree.

Note: *The diameter of a tree is the largest of all shortest-path distances between any two leaves in the tree.*

Putting the above definitions together, we can say that BECDLMST for any N given ground nodes and M given agents, is the Steiner tree spanning all N nodes with at most M edges, such that maximum latency (i.e. diametric latency or latency along the tree diameter), τ as defined in Equation 3.1, is minimized.

Figure 3.5 shows examples of BECDLMSTs to illustrate the above described concept.

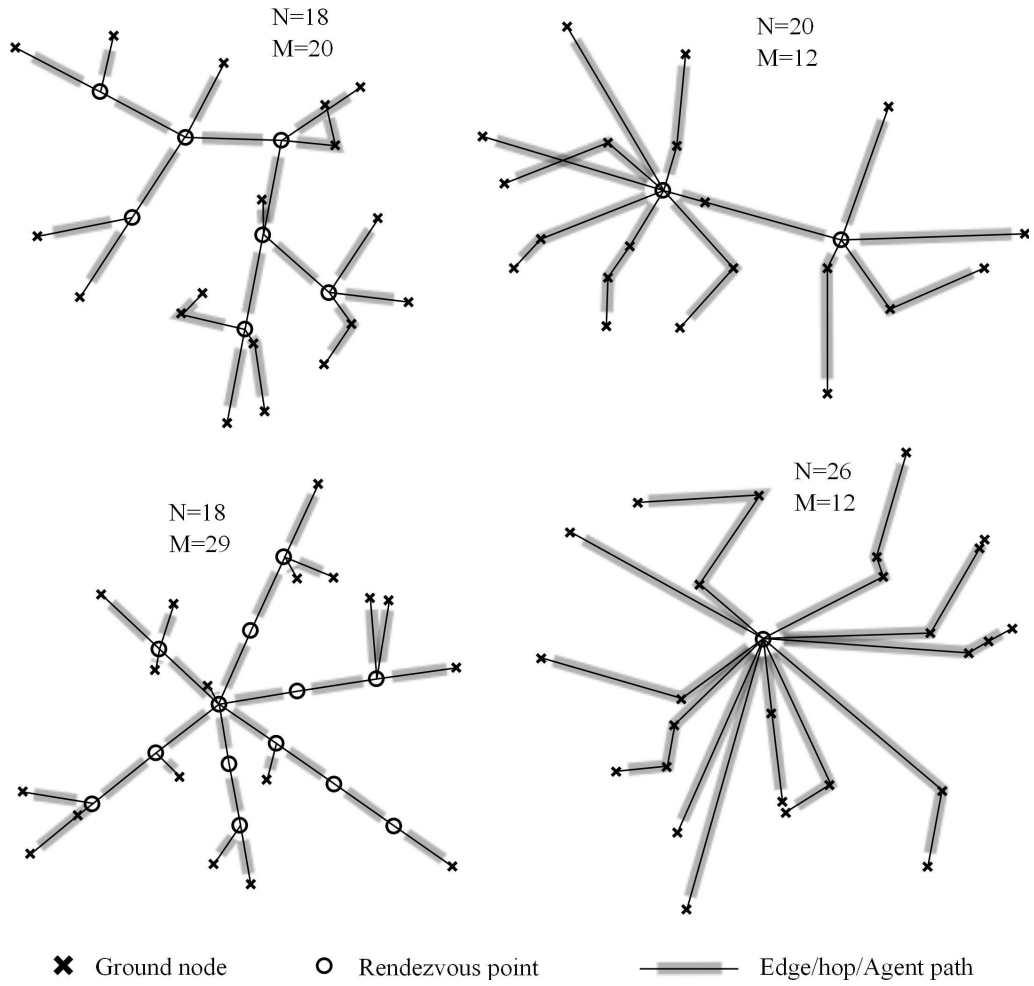


Figure 3.5: BECDLMST for random configurations of varying number of ground nodes and agents

Supporting characteristics of BECDLMST

We now take a look at some of the additional reasons why we propose BECDLMST as the solution structure for **Problem1** described in Chapter 2. Essentially we show that a tree-like structure does not compromise on solution quality when compared to solutions with agent paths containing simple cycles.

Note: *A simple cycle or circuit is a closed path where no vertex or edge is repeated except for the starting and ending vertex.*

1. The first reason is that for any pair of nodes, A and B, the path for packets from A to B, combined with the path for packets from B to A, need not contain simple cycles. The reasoning is that if there was a simple cycle, then the two paths, $A \rightarrow B$ and $B \rightarrow A$ are different and one could possibly be longer than the other. But the maximum latency still depends on the longer of the paths, which is the bottleneck. Therefore, the presence of the simple cycle does not provide an advantage. Below is a formal proof to this claim.

   Claim 3.3.1 *For any pair of nodes, A and B, the path for packets from A to B, combined with the path for packets from B to A, need not contain simple cycles.*

   Proof: Let us assume that the packet originating at node A, destined for node B has to pass through a set of nodes G_{AB} and a set of rendezvous points V_{AB}, both of which could potentially be empty. Along the path A to B, the maximum latency occurs for the packet starting at A and ending at B. The path that minimizes this maximum latency would then be given by the corresponding TSP solution, which we know cannot contain cycles. If the path from A to B were followed in the opposite direction for packets

originating at B and destined for A, the maximum inter-node latency among all pairs in $\{A, B\} \cup G_{AB}$ would remain unaltered at latency of A-B, thus not increasing the maximum latency. □

It follows from Claim 3.3.1 that there is no advantage to the quality of a solution in assigning cyclic agent paths. This supports the fact that agents are assigned acyclic paths that are traversed back and forth in BECDLMST.

2. Now, even if each agent was assigned an acyclic path, the combination of paths assigned to multiple agents, could potentially lead to graphs that have simple cycles. However, our proposed solution is a tree. We now show that using a tree-like solution structure does not compromise on solution quality (in terms of network latency) as compared to those with simple cycles in them.

   Firstly, from [34], we have:

   Theorem 3.3.2 *[Proof in [34]]. For a given set of points, the Steiner tree (i.e. the spanning tree with additional allowed points) that minimizes the diameter of the tree (i.e. the longest path in the tree), is constructed by introducing a single Steiner point at the center of the minimum enclosing circle (MEC), and connecting each point to the Steiner point. The length of the minimum diameter of the Steiner tree is the diameter of the MEC.*

   Claim 3.3.3 *For every solution where the combination of paths assigned to agents contains a simple cycle, an equally good or better solution exists that does not contain simple cycles.*

   Proof: Let us consider a set of k agent paths, $\{A_1$-A_2, A_2-$A_3, \ldots, A_{k-1}$-A_k, A_k-$A_1\}$, that form a simple cycle. Each agent travels back and forth along its assigned path, thus keeping in line with Claim 3.3.1. We argue that replacing the simple cycle with a tree can reduce maximum latency. The tree that minimizes

node-node path length while allowing the addition of rendezvous points, is the minimum diameter Steiner tree (MDST) [34]. According to Theorem 3.3.2, we can build an MDST here by introducing a rendezvous/Steiner point, R at the center of the MEC of $A_1 \ldots A_k$ [34]. The new set of k agent paths would then be $\{A_1\text{-}R, A_2\text{-}R, \ldots, A_k\text{-}R\}$. The longest path then has a length equal to the diameter of the MEC, regardless of k. However, in the simple cycle, the longest path would increase with k. Therefore the best case for the simple cycle would be when $k = 3$, in which case, the maximum path length is given by the longest pairwise distance between A_1, A_2, A_3. According to the classical result from Jung's theorem [35], the longest pairwise distance in any group of points has a lower bound of $\sqrt{3}r_M$, where r_M is the radius of the MEC. This would happen when $A_1A_2A_3$ form an equilateral triangle. The lowest possible maximum latency for the case with a simple cycle is then along one of the sides of the equilateral triangle and given by:

$$\begin{aligned} \text{Simple cycle : Latency, } \tau &= t_w + t_p \\ &= 2\sqrt{3}\tfrac{r_M}{v_{max}} + \sqrt{3}\tfrac{r_M}{v_{max}} \\ &= 3\sqrt{3}\tfrac{r_M}{v_{max}} \end{aligned}$$

The maximum latency in the MDST however is caused by the path of length $2r_M$, passing through R. Using Equation 3.1, the maximum latency for the MDST is then given by:

$$\begin{aligned} \text{MDST : Latency, } \tau &= t_w + t_p \\ &= 2\tfrac{r_M}{v_{max}} + 2\tfrac{r_M}{v_{max}} \\ &= 4\tfrac{r_M}{v_{max}} \end{aligned}$$

We know that $4\frac{r_M}{v_{max}}$ is less than $3\sqrt{3}\frac{r_M}{v_{max}}$. The case of the equilateral triangle (i.e. best case involving a simple cycle) is illustrated in Figure 3.6.

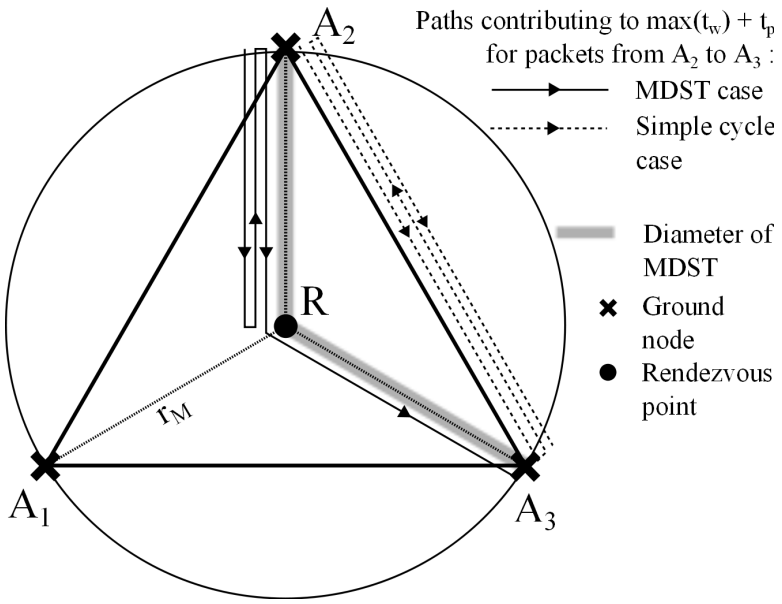


Figure 3.6: Replacing the simple cycle with an MDST in the case of an equilateral triangle

Therefore, even in the best case for the simple cycle, the MDST presents a lower maximum latency. As a result, any occurrence of a simple cycle can be replaced by the corresponding MDST. □

Finding the BECDLMST

The task of **Problem1** is now reduced to finding the BECDLMST for any given configuration of N ground nodes with M agents. In other words we need to construct the Steiner tree spanning all ground nodes by determining rendezvous points (RPs) and corresponding M agent paths (edges), such that network latency, τ, is minimized.

Finding the optimal BECDLMST as such is an NP optimization problem. The decision version of the problem, which we term Dec-BECDLMST, will have to determine whether a Steiner tree with at most M edges can be constructed with an upper bound on n_h (i.e. number of hops on tree diameter) and λ_h (i.e. length of

longest edge/hop in tree). Dec-BECDLMST is similar to the Bounded Diameter Bounded Cost Spanning Tree (BDBCST) problem, that has been proven to be NP complete [34]. Dec-BECDLMST is easy to visualize using the following analogy:

Imagine a bulletin board with N pins stuck on it at various locations. You are given M pieces of thread each of length λ_h. You are allowed to shorten any of the threads by any amount if required. If two threads are to be connected, they can only be connected end to end (i.e. one thread cannot connect to the middle of another thread). The problem then is to determine if it is possible to connect all the pins to each other with the given threads such that the path between no two pins requires more than n_h threads.

Dec-BECDLMST can be reduced from the Travelling Salesman Problem (TSP) as shown below.

Theorem 3.3.4 *Dec-BECDLMST, which is the problem of determining whether a Steiner tree can be built for any given N nodes under the constraints of n_h, λ_h and M, is NP-hard.*

Proof: We use a many-one reduction from the decision version of the Travelling Salesman Problem to show that Dec-BECDLMST is NP hard. Given any instance of the TSP problem with N nodes and L as the limit on path length, an equivalent Dec-BECDLMST instance can be constructed by considering the same N nodes and setting $n_h = 1$, $M = 1$, and $\lambda_h = L$. In other words, any TSP problem is the same as using a single agent to cover all the given nodes with an agent path length smaller than or equal to L. Since every instance of TSP can be reduced to some instance of Dec-BECDLMST, we have proven that Dec-BECDLMST is at least as hard as TSP, and hence NP-hard. □

In actual fact, Dec-BECDLMST is a lot harder than TSP especially when $M \neq 1$. The part of the problem resembling TSP is the one where potential agent paths of length λ_h are sought after. After generating the set of feasible rendezvous points, the problem becomes similar to the Weighted Set Cover problem which is also NP hard. This notion is better understood in the next chapter (Chapter 4) that describes an exact exponential algorithm for finding the optimal BECDLMST. Observing that finding the optimal BECDLMST takes exponential time, we explore efficient methods to approximate the BECDLMST. Our proposed heuristic to find near optimal BECDLMSTs is presented in Chapter 5

3.4 Summary and Contributions

This chapter discussed the solution structure for the agent path design problem described in Chapter 2. Based on the fact that minimal Steiner trees provide the shortest interconnect between any number of points on a plane, we proposed using BECDLMST to provide agent paths that minimize network latency. We described the properties of BECDLMST and showed that using a tree-like structure for agent paths does not compromise on network latency when compared to solutions that contain simple cycles. Finally, we showed that finding the BECDLMST is an NP-hard problem.

The main contribution of this chapter is a novel solution structure for the agent path design problem aimed at minimizing maximum latency in a network (i.e. network latency). It is based on the concept of Steiner trees in geometry.

Chapter 4

Exact exponential algorithm

Having proposed BECDLMST as the solution structure for the problem laid out in Chapter 2, we now take up the task of finding the optimal BECDLMST for any given configuration of N ground nodes with M agents. In the previous chapter, we showed that finding the optimal BECDLMST is an NP-hard problem. In this chapter, we present an exact exponential algorithm to generate the optimal BECDLMST for any given problem setting. Although exponential in nature, the algorithm is designed to prune the solution space as much as possible at every step so as to minimize computation time.

The problem as such is a hard one that requires determining the optimal values for n_h (i.e. number of hops on tree diameter), and λ_h (i.e. length of longest edge/hop in tree), as well as the corresponding tree for any given ground node configuration. Determining valid n_h and λ_h values that minimize network latency, τ, with only M agents is a complex task. We propose an iterative method of updating n_h and λ_h until converging at the minimum τ achievable with M agents. For this purpose, we first design an algorithm to solve Dec-BECDLMST. In other words, the algorithm answers whether a Steiner tree can be built with at most M edges, where the tree

diameter has at most n_h edges and no edge in the entire tree has a length greater than λ_h. Our proposed algorithm to solve Dec-BECDLMST is presented in the next section before detailing how the input to this problem is varied to find a solution to BECDLMST in section 4.2.

4.1 Decision subproblem (Dec-BECDLMST)

To reiterate, Dec-BECDLMST is the problem where we aim to determine whether a Steiner tree with at most M edges can be constructed with an upper bound on n_h (i.e. number of hops on tree diameter) and λ_h (i.e. length of longest edge/hop in tree). The algorithm for Dec-BECDLMST uses the simple idea of generating the set of potential rendezvous points and agent paths, and then checking if there exists a subset that satisfies the requirements of the problem.

Inputs for Dec-BECDLMST are as follows:

M	number of agents
N	number of ground nodes
g_i	position of i^{th} ground node $\forall i = 1, 2, \ldots N$
n_h	number of hops (edges) on maximum latency path
λ_h	maximum hop length in the entire network
MEC	Minimum Enclosing Circle around all ground nodes
r_M	radius of MEC
c_M	center of MEC
v_{max}	maximum speed of any given agent

Note that the n_h and λ_h values received as input are upper bounds and we are trying to determine if a tree can be constructed within these bounds.

4.1.1 Sub-hop-paths

In trying to solve this problem we use the characteristics laid out in Section 3.3. We first introduce the concept of a *sub-hop-path*. A *sub-hop-path* is defined as a path through ground nodes starting at either a ground node or a rendezvous point that has a length $\leq \lambda_h$. A *sub-hop-path* can potentially be assigned to 1 agent. *Sub-hop-paths* that start at ground nodes are referred to as terminal or level 0 *sub-hop-paths*, and form the outermost edges in the Steiner tree. We recall here that an edge in the context of BECDLMST is a path with one end at an intermediate vertex and another end at either an intermediate vertex or a ground node, that passes through zero or more ground nodes without any branches in between. Note that the center of the diameter of the tree is considered to be the root/pole. If number of hops on the tree diameter, n_h, is even, the tree is monopolar with a single rendezvous point considered as the root. If n_h is odd, the tree is dipolar with two roots. In every *sub-hop-path*, the first node is the furthest from the root and the last node, the closest. The level of a *sub-hop-path* is given by the maximum number of rendezvous points along any path starting from this *sub-hop-path* and moving away from the root. The maximum/deepest level possible is $\lfloor \frac{n_h}{2} \rfloor - 1$. We define:

$$\begin{aligned} P^j &= \text{set of } \textit{sub-hop-paths}, p, \text{ at level } j \\ p[k] &= k^{th} \text{ point in the } \textit{sub-hop-path}, p \\ len(p) &= \text{length of } \textit{sub-hop-path}, p \end{aligned}$$

From Theorem 3.3.2, we know that the lower bound on path length of the diameter of the tree is given by the diameter of the MEC. The n_h and λ_h values received as input could be such that $n_h\lambda_h \geq 2r_M$. This would mean that the diameter of the tree does not have to be a straight line. We then define slack of the problem, δ, as follows:

$$\delta = \sqrt{\left(\frac{n_h}{2}\lambda_h\right)^2 - {r_M}^2} \tag{4.1}$$

δ specifies the maximum distance of the root from the center of the MEC. The function f is defined over all $x \in \mathbb{R}^2$ to give the maximum distance between x and any ground node, as follows:

$$f(x) = \max_i \|x - g_i\| \tag{4.2}$$

We then define two functions:
For all points, $x \in \mathbb{R}^2$, and $j \in \mathbb{N}$,

$$lim1(x, j) = \lambda_h \left(\frac{n_h}{2} - j\right) + \delta - \|x - c_M\| \tag{4.3}$$

$$lim2(x, j) = \lambda_h(n_h - j) - f(x) \tag{4.4}$$

The following constraint is then defined to determine the validity of *sub-hop-paths*.

$$\begin{aligned} \forall p \in P^j \quad len(p) \leq \min(\quad & lim1(p[last], j), \\ & lim2(p[last], j), \\ & \lambda_h) \end{aligned} \tag{4.5}$$

Note that j in Equation 4.5 refers to the level of *sub-hop-paths* being considered. The condition in Equation 4.5 represents the necessary limiting condition on the length of a *sub-hop-path* to ensure validity. The condition $len(p) \leq lim1(p[last], j)$, represents the fact that the *sub-hop-path* needs to, at the least, be able to reach the closest possible root in $\frac{n_h}{2}$ hops. The maximum possible offset of the root from c_M is given by δ and hence the closest possible root to $p[last]$ is the point at a distance δ from c_M towards $p[last]$. An illustration of this is shown in Figure 4.1. The second condition $len(p) \leq lim2(p[last], j)$, says that the path length from $p[0]$ to the furthest point from $p[last]$ should not exceed $n_h\lambda_h$. The third condition $len(p) \leq \lambda_h$, is the basic requirement that length of the *sub-hop-path* cannot exceed λ_h.

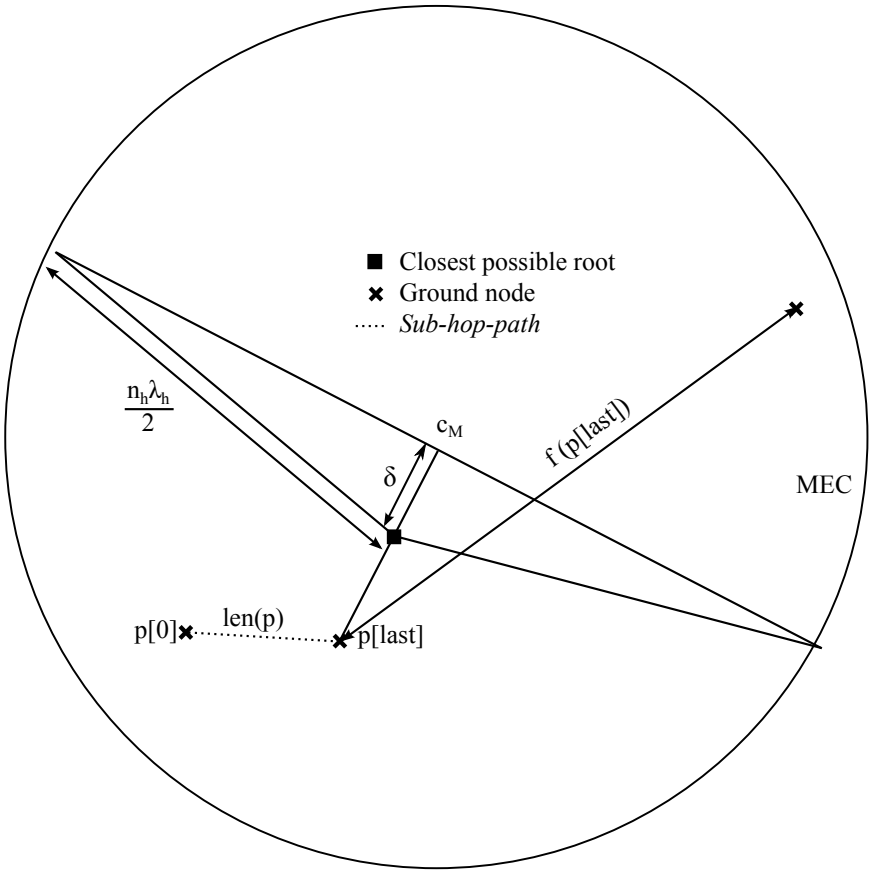


Figure 4.1: Necessary conditions for valid *sub-hop-path*

The first step in the algorithm now is to find all valid terminal *sub-hop-paths*. This is done by generating all possible acyclic paths starting at each ground node, subject to the constraint in Equation 4.5. In trying to generate terminal *sub-hop-paths*, for each path, q, that fails to meet the $lim1$ condition, we take note of the margin of failure and maintain the minimum of all such values. We use the term μ to represent this minimum and express it as follows:

$$\mu = \min_{\substack{q \in \text{tested terminal paths} \\ \text{that failed Equation 4.5}}} len(q) + \|q[last] - c_M\| \tag{4.6}$$

μ is not used in the solution to Dec-BECDLMST. Its usage in finding the optimal BECDLMST is detailed in Section 4.2.

4.1.2 Rendezvous points

Rendezvous points are where agents meet. They are the non-leaf vertices in the Steiner tree. *Sub-hop-paths* are edges in the tree that are connected to each other through rendezvous points. We assign levels to rendezvous points depending on how close they are to the root. The outermost rendezvous points have a level of 0, and this level increases the closer they are to the root, thus making the root, the rendezvous point with the highest level of $\lfloor \frac{n_h}{2} \rfloor - 1$. More formally, the starting point, $p[0]$, of a level k+1 *sub-hop-path* is a level k rendezvous point. We define RP^k as the set of rendezvous points at level k. Two types of rendezvous points are identified:

1. Merging rendezvous point (MRP) : MRPs are vertices in the Steiner tree with more than one child. They are enumerated by considering pairs of *sub-hop-paths* and generating feasible points that can act as parents/ancestors with

each *sub-hop-path* lying on different subtrees of said parent. The points to be considered for each pair of *sub-hop-paths* are the points of intersection of circles centered at $p[last]$ of both *sub-hop-paths*, with radii chosen according to the rendezvous level being filled (discussed below).

2. Linking rendezvous point (LRP) : LRPs are vertices in the Steiner tree with a single child. They connect a *sub-hop-path* of a lower level with a *sub-hop-path* of a higher level. But when generating rendezvous points at level k, we do not have the set of level k+1 *sub-hop-paths* available. So we generate a level k+1 *sub-hop-path* while generating a level k LRP. For the new level k+1 *sub-hop-path*, the starting point $p[0]$ would then be the LRP. The remainder of the *sub-hop-path* given by $p[1]$ to $p[last]$ would have to be ground nodes. The set of paths that could possibly make up $p[1]$ to $p[last]$ is then the set of level 0 *sub-hop-paths*. Formally, if p_1 is a level 0 *sub-hop-path*, and p_2 a level j *sub-hop-path*, the point of intersection of the line joining $p_1[0]$ and $p_2[last]$, with the circle of radius $\lambda_h(1+k-j) - len(p_2)$ centered at $p_2[last]$, is a potential LRP.

We define the operator, $\bowtie$, as follows: If $C(x,r)$ represents the circle of radius r, centered at the point x,

$$C(x,r1) \bowtie C(y,r2) = \text{Set of points of intersection of } C(x,r1) \text{ and } C(y,r2)$$

Algorithm 1 is then applied to populate all the sets of *sub-hop-paths* and rendezvous points. The first seven stages in Figure 4.2 show an example run of the algorithm on a given ground node configuration of 7 nodes, with inputs, $n_h = 4$ and $\lambda_h = \frac{r_M}{2}$, where r_M is the radius of the MEC.

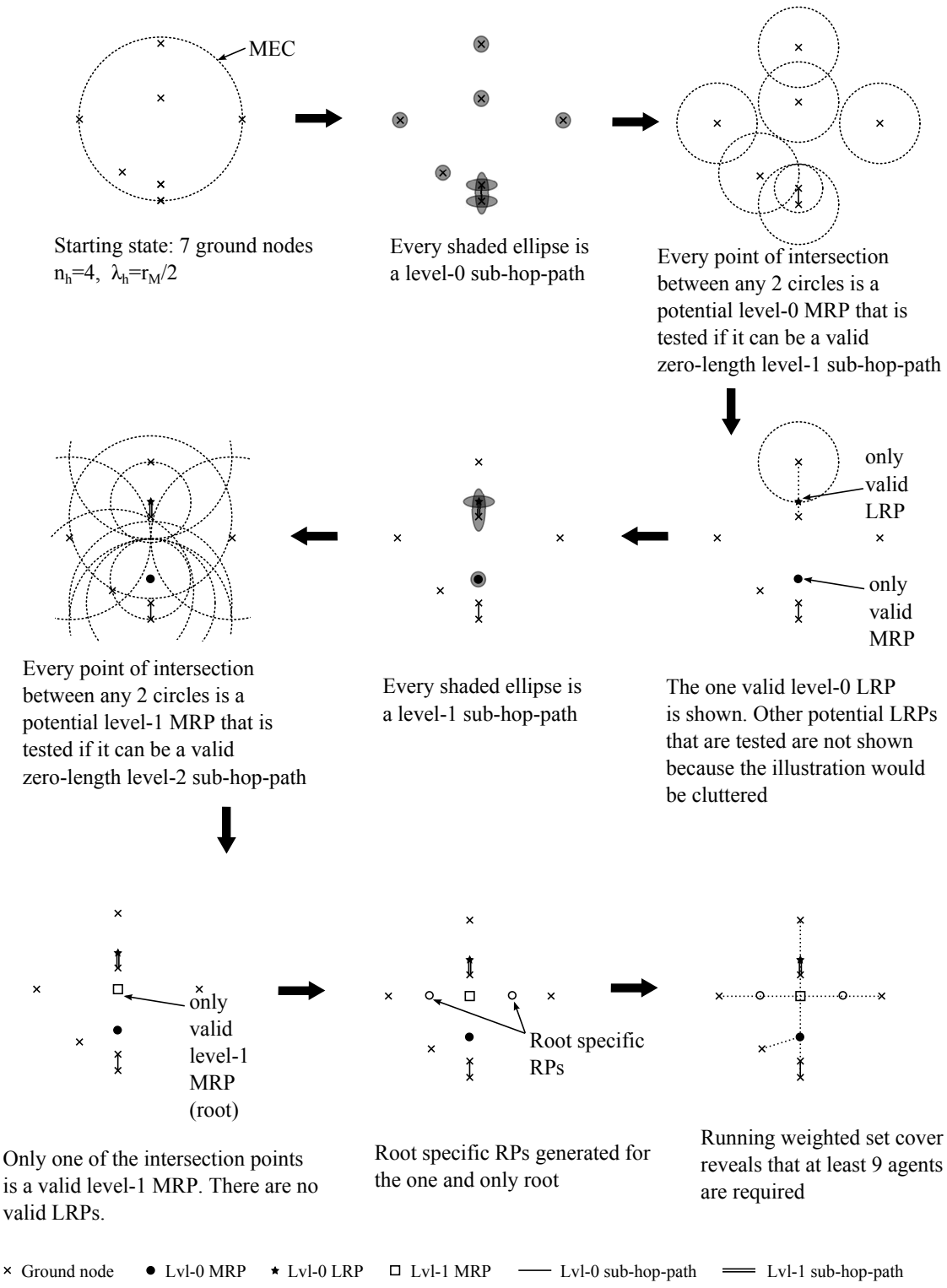


Figure 4.2: Example run of Algorithm 1 and generation of root-specific rendezvous points

Algorithm 1 Populate sub-hop-paths, MRPs and LRPs

for $k = 0$ to $\lfloor \frac{n_h}{2} \rfloor - 1$ **do** {Note: k is level}

#Generate level k sub-hop-paths
if $k = 0$ **then**
for all $g \in$ Ground nodes **do**
Generate acyclic paths starting at g, running through ground nodes, subject to constraint in Equation 4.5 and add them to P^0
end for
else
for all $r \in RP^{k-1}$ **do**
Generate acyclic paths starting at r, running through ground nodes, subject to constraint in Equation 4.5 and add them to P^k
end for
end if

#Generate level k MRPs
for all $p_1 \in \bigcup_{0 \le i \le k} P^i, p_2 \in \bigcup_{0 \le j \le k} P^j, c \in \{i \dots k\}$ **do**
if $p_1 \neq p_2$ **then**
$r_1 \leftarrow \lambda_h(1 + k - c) - len(p_1)$
$r_2 \leftarrow \lambda_h(1 + k - j) - len(p_2)$
for all $x \in C(p_1[last], r_1) \bowtie C(p_2[last], r_2)$ **do**
if $lim1(x, k+1) \ge 0 \wedge lim2(x, k+1) \ge 0$ **then**
Add x to RP^k
end if
end for
end if
end for

#Generate level k LRPs
for all $p_1 \in P^0, p_2 \in \bigcup_{0 \le j \le k} P^j$ **do**
if $p_1 \neq p_2$ **then**
$x \leftarrow p_2[last] + \frac{\lambda_h(1+k-j) - len(p_2)}{\|p1[0] - p2[last]\|}(p_1[0] - p_2[last])$
if $lim1(x, k+1) \ge 0 \wedge lim2(x, k+1) \ge 0$ **then**
Add x to RP^k
end if
end if
end for

end for

For MRP generation in Algorithm 1, r_2, the radius for the circle centered at $p_2[last]$ is set such that the level of the rendezvous point being generated is maintained. For example, if $k = 2$ and p_2 was a level 0 *sub-hop-path*, the new rendezvous point should be exactly 2 hops away from the immediate next rendezvous point. When p_2 ensures the level of the new rendezvous point, p_1 is free to have a radius given by any number of hops less than or equal to the number of hops required for p_1 to ensure level k for the new rendezvous point. Every intersection point thus generated is subject to limiting conditions similar to those in Equation 4.5 by viewing the new level k rendezvous point as a zero-length, level k+1 *sub-hop-path*.

Claim 4.1.1 *In algorithm 1, considering $C(p_1[last], r_1)$ and $C(p_2[last], r_2)$, although any point in the overlapping region of the circles is a potential rendezvous point, it is sufficient to test-and-add just the points of intersection as MRPs.*

Proof: Let us say the final optimal BECDLMST required an MRP, x, to connect a set of *sub-hop-paths* $\{p_{a1}, p_{a2} \ldots p_{ab}\}$. Let rendezvous point, y, be the immediate parent of x in the tree. The requirement is therefore that x has to lie within the appropriate ranges of the last nodes of each of $\{p_{a1} \ldots p_{az}\}$ as well as y. So x lies in the region of overlap of $(z + 1)$ circles, one for each *sub-hop-path* served and one for y. Within this region, x can be moved anywhere without a loss in solution quality. Every corner in this region is definitely a point of intersection between some pair of circles. Since we consider every pair of *sub-hop-paths*, an intersection point given by at least one such pair is bound to satisfy x. □

In the context of the above proof, MRPs would have taken care of the first z circles. The $(z + 1)^{th}$ circle is indirectly considered in the case of LRPs as well as in the next section. Figure 4.3 illustrates the algorithm thus far on another sample

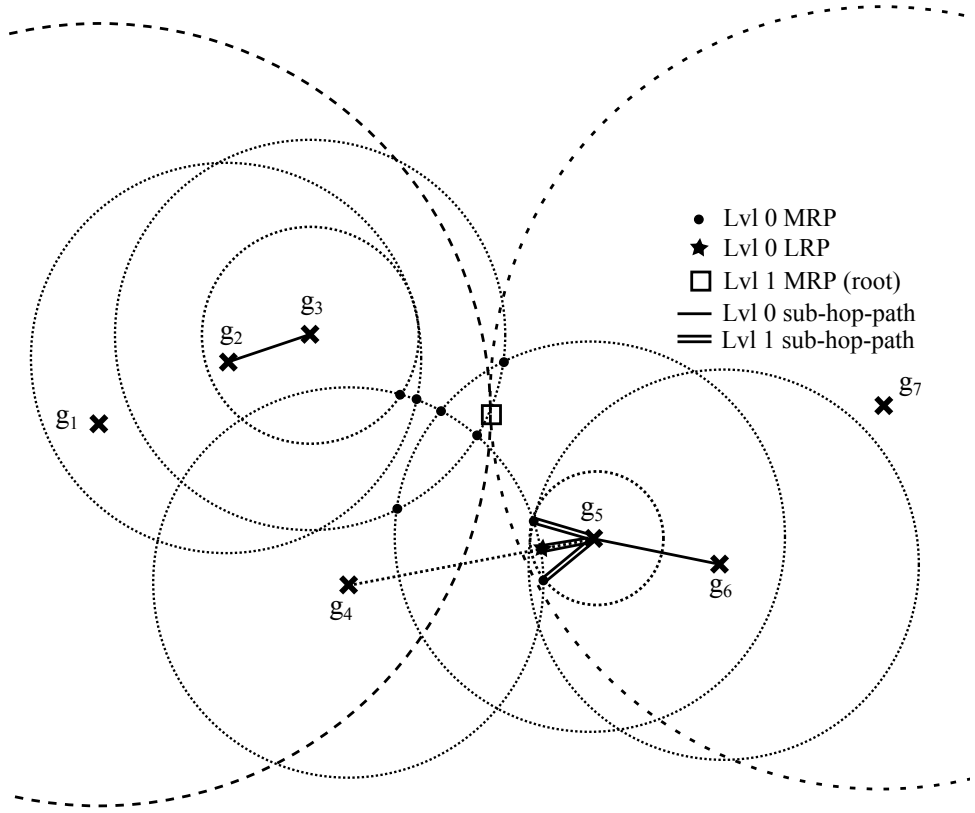


Figure 4.3: Rendezvous points for a sample node configuration ($n_h = 4, \lambda_h = \frac{r_M}{4}$) Notes: 1) The circles drawn show all the circles that are intersected with each other to generate RPs. 2) Notice that all level 1 sub-hop-paths start from a level 0 RP. 3) The LRP above is at the intersection of the line joining g_4 and g_5 and the circle centered at g_4. 3) Only valid rendezvous points and sub-hop-paths are shown.

configuration of ground nodes. All valid rendezvous points generated using the algorithm up to this point are shown.

4.1.3 Root-specific rendezvous points

Let,

$$\gamma = \lfloor \frac{n_h}{2} \rfloor - 1$$

Now RP^γ holds the set of feasible roots for the Steiner tree. If n_h was even, the tree would be monopolar. In fact if $n_h \lambda_h = 2r_M$, we are assured that $|RP^\gamma| = 1$. On the other hand, if n_h was odd, the tree would be dipolar, with separation between

the roots $\leq \lambda_h$.

We define the set $\mathbb{S}_{roots}$ as follows:

$$\begin{aligned}
&\text{If} \quad n_h \text{ is even} \\
&\quad \mathbb{S}_{roots} = \{S: \quad S \subseteq RP^{\gamma} \wedge |S| = 1\} \\
&\text{If} \quad n_h \text{ is odd} \\
&\quad \mathbb{S}_{roots} = \{S: \quad S \subseteq RP^{\gamma} \wedge |S| = 2 \\
&\qquad\qquad\qquad \wedge (x \in S \wedge y \in S \rightarrow \|x - y\| \leq \lambda_h)\}
\end{aligned} \tag{4.7}$$

The next step is to determine if any of the elements in $\mathbb{S}_{roots}$ can be used to achieve the requirement posed by n_h and λ_h using just M agents. We consider each element, $\Gamma \in \mathbb{S}_{roots}$, one by one and terminate when the answer to the decision problem (Dec-BECDLMST) is found.

We know that members of Γ must have originated as the MRP for some two *sub-hop-paths*. This means that the chosen rendezvous points in Γ might not have links to some of the other *sub-hop-paths*. We do a simple breadth-first traversal starting from the rendezvous points in Γ and moving 1 hop at a time to generate a connected graph of *sub-hop-paths*, G_{Γ}. Rendezvous points must now be introduced to link the unlinked *sub-hop-paths* if possible. In order to have the least number of agents, the unlinked *sub-hop-paths* must be linked to the existing graph using the least number of hops. After graph G_{Γ} is generated, for each unlinked *sub-hop-path*, p_1, we find the closest *sub-hop-path*, p_2, in G_{Γ} (determined by $\|p_1[last] - p_2[0]\|$) that satisfies the following condition:

$$\begin{aligned}
&p_1 \in P^i \wedge p_2 \in P^j \\
\wedge \quad &j > i \ \wedge \ len(p_1) + \|p_1[last] - p_2[0]\| \leq \lambda_h(j - i)
\end{aligned} \tag{4.8}$$

If no such p_2 exists, p_1 is just ignored and termed unreachable. The ground nodes covered by p_1 will definitely still be reachable through other *sub-hop-paths*. If p_2 is

found, $p_1[last]$ and $p_2[0]$ are connected by introducing a series of rendezvous points along the line connecting them. For each new rendezvous point, the corresponding zero-length *sub-hop-path* starting at that rendezvous path, is added to the appropriate level *sub-hop-path* set. These rendezvous points are however temporary and not used once Γ is changed. Figure 4.4 illustrates root-specific rendezvous points introduced in continuation from Figure 4.3. The eighth stage in Figure 4.2 is also an illustration of what root-specific rendezvous points are.

4.1.4 Set Cover for any root, Γ

Having enumerated the possible rendezvous points, we now need to determine if a combination of less than M connected *sub-hop-paths* exists, such that all ground nodes are covered. Solving this portion of the problem could potentially use a branch and bound algorithm that starts at the members of Γ and branches out towards the lower level *sub-hop-paths*. However, the nature of the problem would make that expensive because the ground nodes are, for the most part, on terminal *sub-hop-paths*. The benefit of branch and bound would be lost. Instead we choose to work from the leaves towards the root. We define a weighted node cover subset (WNCS), as a set of ground nodes with a corresponding integer weight. The integer weight is used to represent the number of agents required. We start by creating one single-weighted WNCS per reachable *sub-hop-path* containing the ground nodes covered by the corresponding *sub-hop-path*.

$$\forall p \in P^0 \quad \text{WNCS}(p) = \{p[0], p[1] \dots p[last]\}_1$$

$$\forall j \in \{1 \dots \gamma\} \quad \forall p \in P^j \quad \text{WNCS}(p) = \{p[1] \dots p[last]\}_1$$

Note: WNCS weight represented using right subscript

Next we define the bucket, B, of a *sub-hop-path* as the set of possible WNCSs it provides to the next *sub-hop-path* along its route towards the root.

$$\forall p \in P^0 \quad B(p) = \{\varnothing, \text{WNCS}(p)\}$$

The set of immediately reachable lower level *sub-hop-paths* for *sub-hop-path*, p, is represented as $children(p)$. Buckets for each reachable *sub-hop-path*, p, are generated using the buckets of $children(p)$. We define the commutative, associative, binary operator $\uplus$, on buckets, as follows:

$$\begin{aligned} B_1 \uplus B_2 \quad = \quad & \{S_w \colon S_w = S_{1w_1} \cup S_{2w_2} \text{ and } w = w_1 + w_2 \\ & \text{where } S_1 \in B_1, S_2 \in B_2\} \end{aligned}$$

Using the $\uplus$ operator, we trickle the WNCSs from the terminal *sub-hop-paths* towards the root. Let,

$$\hat{B}(p) = \{(\text{WNCS}(p) \cup S_w)_{w+1} : S_w \in \uplus_{q \in children(p)} B(q)\}$$

Then $B(p)$ is the largest possible subset of $\hat{B}(p)$ subject to the condition:

$$\begin{aligned} S_1, S_2 \in \hat{B}(p) \quad \wedge \quad S_1 \subseteq S_2 \quad \wedge \quad \text{weight}(S_1) \geq \text{weight}(S_2) \\ \rightarrow S_1 \notin B(p) \end{aligned} \tag{4.9}$$

The rule in Equation 4.9 ensures that all inferior solutions are weeded out early. Only the best possible buckets are passed from one *sub-hop-path* to the next. At the end, all the children of the root, $children(\Gamma)$, would present their buckets to the root. At this point, the problem becomes the Weighted Set Cover decision problem. The problem can be stated as:

Given the subsets contained in buckets $B(q) \forall q \in children(\Gamma)$, is there a combination of subsets with not more than one occurring from each bucket, such that every

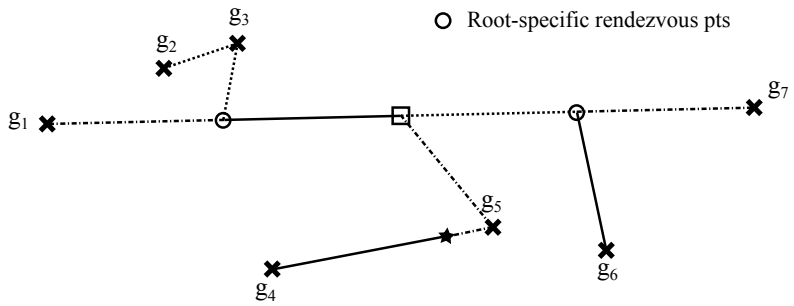


Figure 4.4: Agent paths with $M = 8$ for same node configuration as in Figure 4.3 $(n_h = 4, \lambda_h = \frac{r_M}{4})$

ground node occurs in at least one of the chosen subsets, and the sum of weights of all chosen subsets does not exceed M, the number of agents.

The above is a slightly restrictive version of Weighted Set Cover with a smaller search space, owing to the condition that only one subset can be chosen from each bucket. A standard branch and bound algorithm is applied here to obtain the answer. If the answer is no, the steps in section 4.1.3 and 4.1.4 are repeated for the next $\Gamma \in \mathbb{S}_{roots}$. If all elements in $\mathbb{S}_{roots}$ are exhausted and the answer is still no, the answer to Dec-BECDLMST is returned as no.

The agent paths determined by one of the set covers for the node configuration used in Figure 4.3 is shown in Figure 4.4.

4.2 Determining the optimal BECDLMST

We now describe the algorithm that uses the solution to **Problem1** in an iterative fashion to arrive at the optimal BECDLMST. From Theorem 3.3.2, we know that the lower bound on $n_h\lambda_h$ is $2r_M$, where n_h is the number of hops on the tree diameter and λ_h is the length of the longest edge in the tree. The plot of maximum latency, τ, against λ_h for different values of n_h is shown in Figure 4.5. It is based on the equation: $\tau = \frac{1}{v_{max}}(n_h\lambda_h + 2\lambda_h)$

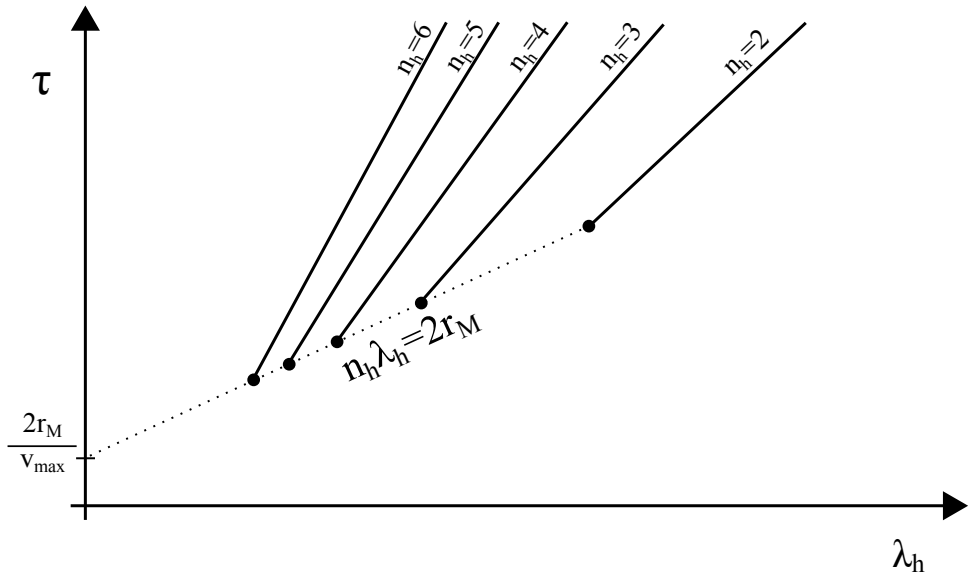


Figure 4.5: Plot of τ against λ_h for different values of n_h

If we fixed $n_h\lambda_h = 2r_M$, increasing n_h reduces τ but increases the number of agents required. The strategy we employ is: while maintaining λ_h at the valid minimum, increase n_h at every iteration, until Dec-BECDLMST returns a no. Once this happens, we fix n_h and increase λ_h as long as the corresponding τ value does not exceed the lowest possible with $n_h - 1$. An example run might look like this when visualized in Figure 4.5: Assume we start at $n_h = 2$, i.e. the lowest point on the line for $n_h = 2$. We then jump to the lowest point on the line for $n_h = 3$ and so on until Dec-BECDLMST returns a no. Assume this happens at $n_h = 5$. We then start climbing along the line for $n_h = 5$ as long as τ stays below the lowest point on $n_h = 4$. Along the climb, the first λ_h that gets a yes from Dec-BECDLMST is then our solution. If no valid λ_h is found, the solution is the lowest point for $n_h = 4$.

To determine $n_h[0]$, i.e. the value of n_h in the first iteration, we consider the MDST built with a single rendezvous point at the center of the MEC. On this tree, it would be possible to assign $\left\lfloor \frac{M}{N} \right\rfloor$ number of agents on each edge, effectively making $n_h = 2\left\lfloor \frac{M}{N} \right\rfloor$ easily satisfiable with M agents. As a result, the starting

value for n_h is given by:

$$n_h[0] = \max\left(2, 2\left\lfloor\frac{M}{N}\right\rfloor + 1\right) \tag{4.10}$$

The above expression also sets the minimum limit for n_h to 2 because if $M = 1$, the solution is just the TSP tour. Our algorithm is meant for $M > 1$, in which case n_h has to be at least 2.

As for λ_h, the strategy is to always use the lowest possible value corresponding to n_h. It is only when the algorithm to Dec-BECDLMST returns a no, that we fix n_h and increase λ_h. Determining the minimum hop length for n_h, $\lambda_{h_{min}}^{n_h}$, depends on whether n_h is odd or even. When n_h is even, the root is a single rendezvous point. Considering the property of MECs stating that the arc length between no two consecutive nodes on the MEC can exceed half the length of the circumference, the optimal location of the root that minimizes the maximum path length between any pair of outliers, is the center of the MEC. As a result, the smallest possible $\lambda_{h_{min}}^{n_h}$ for an even n_h is given by $\frac{2r_M}{n_h}$.

In the case of odd n_h, the root is a pair of rendezvous points located such that distance between them is $\leq \lambda_{h_{min}}^{n_h}$. So we need to partition the outliers such that one partition connects to one of the root nodes and the other partition connects to the other root node. To reduce the distance between a given root node and the partition that connects to it, we identify the 2 largest arcs on the circumference, each caused by any 2 consecutive nodes. Let us label the nodes responsible for the first arc, x_1 and x_2, and the nodes responsible for the second arc, x_3 and x_4 while moving along the same direction around the MEC (refer to Figure 4.6). We want to find $x_{\Gamma[1]}$ and $x_{\Gamma[2]}$ such that λ_h is minimized and the hop constraints are met.

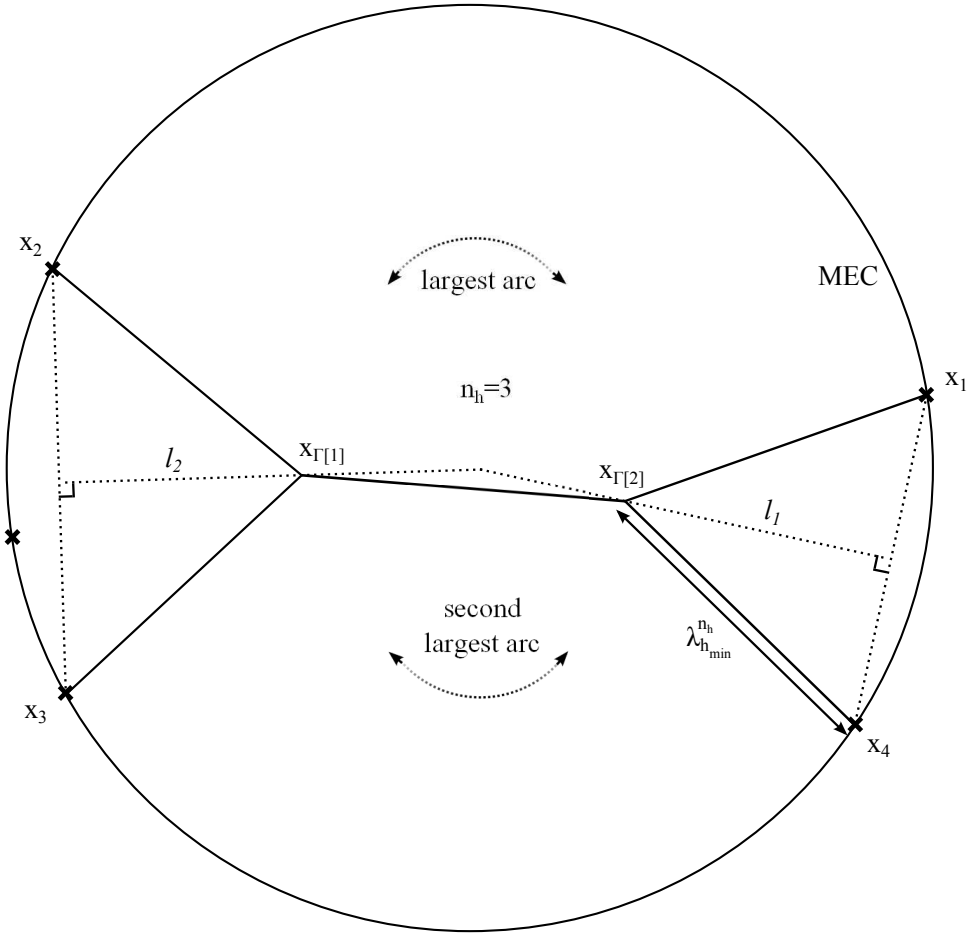


Figure 4.6: $\lambda_{h_{min}}^{n_h}$ when n_h is odd (above: $n_h = 3$)

This is treated as a simple constraint optimization problem described as follows:

$$\begin{aligned} &\text{find } x_{\gamma[1]} \text{ and } x_{\gamma[2]} \text{ to minimize } \lambda_h \\ &\text{subject to } \left\lfloor \tfrac{n_h}{2} \right\rfloor \lambda_h &&= |x_1 - x_{\gamma[1]}| = |x_4 - x_{\gamma[1]}| \\ &&&= |x_2 - x_{\gamma[2]}| = |x_3 - x_{\gamma[2]}| \\ &\lambda_h &&\geq |x_{\gamma[1]} - x_{\gamma[2]}| \end{aligned}$$

Graphically, finding the solution can be viewed as starting from the midpoints of $x_1 \rightarrow x_4$ and $x_2 \rightarrow x_3$ and moving towards each other along lines l_1 and l_2 (refer to figure 4.6), while maintaining the equality constraint, until the inequality constraint is met. x_1 and x_4 could possibly be the same point. Similarly, x_2 and x_3 could possibly be the same point.

As we increase the value of n_h at each iteration, the corresponding $\lambda_{h_{min}}^{n_h}$ is computed and presented as input to Dec-BECDLMST. At the first instance that Dec-BECDLMST returns a no, n_h is fixed as n_{h_f} and λ_h is increased. The only way the number of agents required can be decreased with the same maximum number of hops is by increasing the number of ground nodes handled by some agent in order to render at least one other agent useless and thus eliminate it. So, at least one of the terminal *sub-hop-paths* must be increased in size by 1. In other words, we have to increase the slack, δ, by increasing λ_h to incorporate the path that failed to qualify as a terminal *sub-hop-path* by the smallest margin. This is where we use the term μ that was computed in Equation 4.6. Using equations 4.1, 4.3, 4.5, and 4.6, we derive an expression for the new hop length:

$$\lambda_{h_{new}}^{n_{h_f}} = \frac{\mu^2 + r_M^2}{n_{h_f}\mu} \tag{4.11}$$

$\lambda_{h_{new}}^{n_{h_f}}$ and n_{h_f} are now used as input to the algorithm for Dec-BECDLMST. If the answer is a no, the process is repeated with the new value for μ obtained from the

last run. Through every such iteration, $\lambda_{h_{new}}^{n_{h_f}}$ is consistently increased as long as the following condition is satisfied:

$$\lambda_{h_{new}}^{n_{h_f}} < \frac{2 + n_{h_f} - 1}{2 + n_{h_f}} \lambda_{h_{min}}^{(n_{h_f}-1)} \tag{4.12}$$

In other words, the latency being tested, should not be higher than the one that was achieved using $n_{h_f} - 1$ hops. If Dec-BECDLMST returns a yes before the above condition fails, our final optimal BECDLMST is held by the subsets chosen by Weighted Set Cover in the last run. If the condition fails before Dec-BECDLMST returns a yes, the final optimal BECDLMST is held by the subsets chosen by the algorithm with inputs, $n_{h_f} - 1$ and $\lambda_{h_{min}}^{n_{h_f}-1}$.

4.3 Correctness and Complexity

The correctness of the algorithm lies in its exhaustive nature. Section 4.2 describes how the values of n_h and λ_h are varied to reach the lowest valid point on the plot in Figure 4.5. Since M is finite, there will be a value for n_h that causes Dec-BECDLMST to return a no. Subsequently, λ_h is incremented every iteration. Since λ_h has an upper bound (Equation 4.12), the algorithm will terminate with an answer in finite time. The solution to Dec-BECDLMST ensures that all valid options are explored before concluding. Sub-hop-paths are generated in a fully exhaustive manner with a validity check ensuring only feasible paths are explored. Rendezvous points are categorized as having one child or more than one child, which is exhaustive. Every possible pair of sub-hop-paths is considered to generate rendezvous points. The proof to Claim 4.1.1 shows how no potentially optimal rendezvous points are missed. As a result, the complete set of potential agent paths is presented to the Weighted Set Cover algorithm before deciding yes or no.

Given that the Weighted Set Cover algorithm is used as a subroutine, the complexity of the algorithm is definitely exponential. However, following the discussion in Section 3.3, only those solutions that conform to the tree structure are searched. It is a huge improvement over any path planning approach that does not use this property. We do a quick complexity analysis of each step to give a fuller picture. If n, is the number of ground nodes, the first step of finding the MEC has an $O(n)$ algorithm [36]. Finding the *sub-hop-paths* depends on how clustered the ground nodes are. If k is the expected average distance to nearest neighbor in a random distribution of points within the MEC, the complexity of determining *sub-hop-paths* is nothing but $O(n^{\frac{\lambda_h}{k}})$. Although k is not a constant, the exponent for n is severely limited by λ_h. Determining rendezvous points does a pair-wise comparison of nodes repeatedly for each hop level. Thus complexity for this step can be written as $O(n^{2^{n_h}})$ where n_h is constant for one instance of Dec-BECDLMST. For m agents, the maximum value for n_h is $O(logm)$ because n_h is proportional to the depth of the tree. The worst case complexity for determining rendezvous points can be rewritten as $O(n^m)$. However, the empirical results of algorithm run-time in table 4.2 show that the rate of increase in time is much smaller owing to the constraints that prune the search space at each step of generating rendezvous points. Finally, generating the subset list for Weighted Set Cover, starts from the leaves of the tree and heads towards the root, looking at each tree-node once. As a result, the complexity for this step is linear in the number of rendezvous points and can be expressed to be the same as rendezvous point generation, $O(n^m)$. Although the above analysis provides the worst case complexities, the strict constraints applied at every stage tend to keep the problem size in check.

4.4 Results

We run our algorithm on random ground node configurations for varying values of N and M. Table 4.1 shows the normalized average and standard deviation values for τ (the maximum pairwaise latency) achieved over multiple runs of different configurations for each (N, M) scenario. For each case of N, 50 different random node configurations are generated and for each such configuration, different number of agents (M) are applied. As such, values for N run through 3, 5, 10, 20, and 30 while the number of agents applied are varied between 3, 5, 10, 20, 30 and 40. Note that the values for τ 4.1 are normalized and presented as coefficients of $\frac{r_M}{v_{max}}$.

Table 4.1: Normalized network latency, τ, for various N, M with comparison against latency achieved by FRA [23]

	M=3			M=5		
N	Normalized τ_{FRA}	Normalized $\tau_{BECDLMST}$	Percent improvement	Normalized τ_{FRA}	Normalized $\tau_{BECDLMST}$	Percent improvement
3	17.05±5.06	4.00±0.00	76.53	-	3.23±0.34	-
5	13.71±2.77	5.78±1.33	57.84	13.38±2.79	3.66±0.42	72.65
10	13.51±1.53	8.84±2.29	34.57	12.01±1.59	4.65±0.78	61.28
20	15.97±1.50	12.14±3.92	23.98	13.60±1.50	7.19±1.31	47.13
30	21.65±1.54	18.48±7.20	14.64	15.71±1.54	8.12±2.02	48.31
	M=10			M=20		
N	Normalized τ_{FRA}	Normalized $\tau_{BECDLMST}$	Percent improvement	Normalized τ_{FRA}	Normalized $\tau_{BECDLMST}$	Percent improvement
3	-	2.60±0.15	-	-	2.32±0.08	-
5	-	2.81±0.16	-	-	2.37±0.09	-
10	11.51±1.93	3.44±0.38	70.11	-	2.70±0.15	-
20	10.93±1.66	4.06±0.43	62.85	10.35±1.74	3.17±0.40	69.37
30	11.61±1.31	5.62±0.88	51.59	10.36±1.83	3.88±0.56	62.55
	M=30			M=40		
N	Normalized τ_{FRA}	Normalized $\tau_{BECDLMST}$	Percent improvement	Normalized τ_{FRA}	Normalized $\tau_{BECDLMST}$	Percent improvement
3	-	2.11±0.06	-	-	2.07±0.03	-
5	-	2.18±0.08	-	-	2.12±0.06	-
10	-	2.24±0.10	-	-	2.19±0.08	-
20	-	-	-	-	-	-
30	10.36±1.83	-	-	-	-	-

We observe that τ decreases with an increase in number of agents, but after a certain point, addition of agents makes very little difference. This phenomenon

is caused by the fact that regardless of the number of agents, the distance along the tree diameter has to be travelled. We know from Theorem 3.3.2 that the lower bound on the tree diameter is given by the diameter of the MEC. Therefore τ values cannot go below $2\frac{r_M}{v_{max}}$. As a result, the table shows values that tend asymptotically towards 2 with an increase in M. In actual fact, with an extremely huge number of agents, our initial assumptions on negligible communication range and transmission time would cease to hold. However, we consider sparse networks where communication ranges can safely be assumed to be negligible compared to inter-node distances. We also observe that for any given number of agents, M, the latency τ increases with an increase in N, but the rate of increase is higher when M is lower.

In Table 4.1, we also present a comparison between network latencies achieved by BECDLMST against those achieved by FRA. As discussed earlier, FRA can be cited as the algorithm that achieves the best performance among existing works [Refer to Section 3.2 for a discussion on related work]. Our comparison shows that the optimal BECDLMST, albeit taking exponential time to compute, generates solutions that far outperform FRA. It is important to note that unlike BECDLMST, FRA is only capable of handling problem situations with $N > M$. The results show an improvement in performance up to 76% especially for smaller values of N. We notice that the difference in performance drops as the number of ground nodes increases mainly for small values of M (number of agents). This behavior can be attributed to the fact that when M is small, the number of rendezvous points is very limited. Network latency then mainly depends on the *sub-hop-paths*, which will have to be shortest paths through the nodes they serve. As a result, for low M and high N situations, the advantage of strategic placement of rendezvous points is minimized. Hence performance of FRA gets closer to BECDLMST as N increases for small values of M.

Table 4.2: Average run-times of the exact exponential algorithm for various N, M pairs

N	M=3	M=5	M=10	M=20	M=30	M=40
3	~ 0	~ 0	~ 0	~ 0	~ 0	~ 0
5	~ 0	~ 0	~ 0	~ 0	2	4
10	~ 0	~ 0	2	67	357	982
20	2	9	294	2216	-	-
30	7	42	592	4291	-	-

Table 4.2 now presents the execution time taken (in seconds) to find the BECDLMST in each experiment. The algorithm run-times shown are averages over multiple runs of different node configurations on the same system equipped with a 2.33GHz Core 2 Duo processor. We observe that the algorithm run-time increases exponentially with increase in N. The result adheres to our prior knowledge that WEIGHTED SET COVER is a proven NP-complete problem with an exponential complexity. As a result we omit the prohibitively long cases of high N and high M. We note that the algorithm run-time increases at a higher rate with respect to N as opposed to M. The values indicate that in actual fact N has a stronger effect on the algorithm complexity than M.

Figure 4.7 provides a visual comparison of paths generated by BECDLMST, FRA and a single route solution (SRT). We present a randomly generated ground node configuration with 20 nodes and compare the paths generated by each algorithm for the same configuration. With a single route solution, the TSP tour would have amounted to a final maximum latency of $7.72r_M$ including waiting time as shown in Figure 4.7(a). Figure 4.7(b) on the other hand shows how FRA[23] manages to cap maximum latency at $5.152r_M$. FRA uses a grid-based approach to determine contact points and synchronizes using the longest path between consecutive contact points. We let $M = 12$ to support their algorithm. Finally, the set of agent paths generated by BECDLMST is shown in Figure 4.7(c). As described in Section 3.3, each agent moves back and forth along its assigned path. Agent motions are synchronized with respect to the rendezvous points they visit. For every RP, at

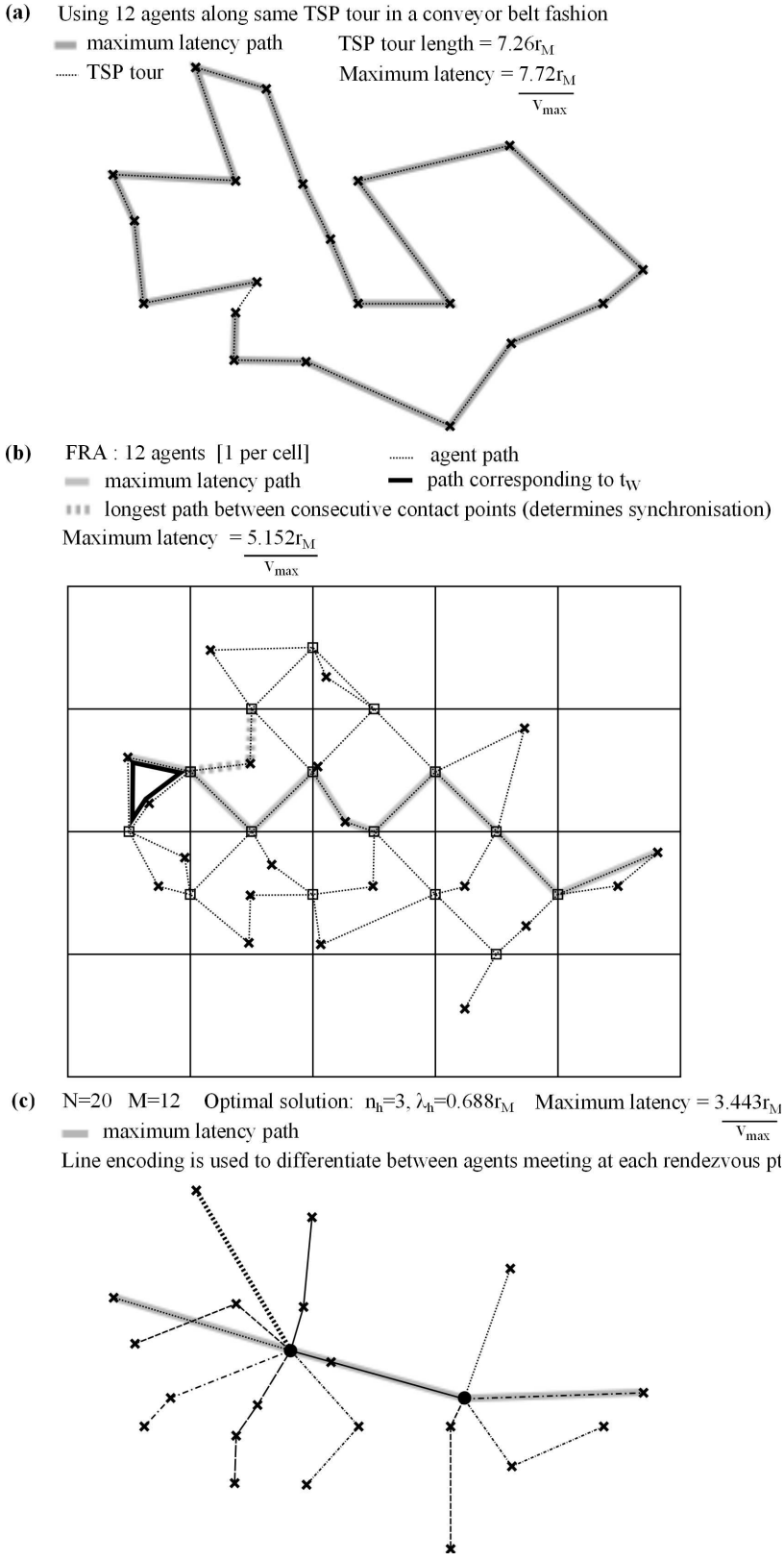


Figure 4.7: Agent paths as generated by: (a) an SRT solution (b) FRA [23] (c) Exact exponential algorithm for BECDLMST, for a random configuration of 20 ground nodes with 12 available agents

any given time, all agents that meet at that RP, are either moving towards it or away from it. M was limited to 12 so as to use no more agents than that required by FRA. The optimal solution gives us a 3-hop tree diameter with a maximum hop length of $0.688r_M$ and achieves a maximum latency of $3.443r_M$. The example here illustrates what the final set of agent paths look like in our solution and show how strategically picking the rendezvous points can lead to lower latency achieving agent paths.

4.5 Summary and Contributions

We presented an exact exponential algorithm to determine the optimal BECDLMST that was described in Section 3.3. The paths generated by the algorithm, showed how given N ground nodes and M agents, strategically placing rendezvous points can lower the worst case latency. The algorithm enumerated the feasible rendezvous points and constructed a set of paths to turn it into the Weighted Set Cover problem. The exponential nature of the algorithm comes as a limitation but the paths generated provide much lower latencies than existing methods. Empirical results showed that BECDLMST provided network latencies that were considerably lower (up to 76%) than those by the existing best solution of FRA. In the next chapter we shall offset the limitation of computational complexity by presenting an efficient heuristic to approximate the BECDLMST.

The main contribution of this chapter is the algorithm to derive the optimal BECDLMST for any given ground node configuration with M agents. It provides completeness to the previous chapter which proposes BECDLMST, by presenting a method to find it.

Chapter 5

Near-optimal efficient heuristic

In this chapter we present an anytime heuristic to find a near optimal solution to BECDLMST. Considering how the exact exponential algorithm can be infeasible for large datasets, we turn to methods capable of producing near-optimal solutions but with much faster computation times. Such optimization techniques are commonly referred to as heuristic algorithms. In order to devise an efficient approximation algorithm for BECDLMST, we first look at the standard meta-heuristic algorithms, i.e. heuristics that do not utilize the specific structure of the problem at hand. In the following section we present a survey of few such algorithms and study their applicability to the problem at hand. We finally use ideas from Particle Swarm Optimization and devise our own heuristic that is tailored to BECDLMST.

5.1 Survey of classical metaheuristics and their applicability

5.1.1 Simulated annealing

Simulated annealing (SA) is a generic probabilistic metaheuristic algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space [37]. It starts with a solution, x, to the problem at hand. This solution is then assigned a value using a scoring function denoted by $f(x)$. Subsequently, one solution in the *neighbourhood*, $N(x)$, of x is evaluated. If the new solution, $\bar{x}$, is better than x, it is chosen as the current solution and the next iteration is commenced. However if $\bar{x}$ happens to be worse than x, then $\bar{x}$ is chosen with a probability of $P(f(x), f(\bar{x}), T)$, where T stands for temperature. Temperature here is a value that consistently decreases in each iteration and represents the idea of metal cooling down in the smithing process (hence the annealing part of the name). The probability P generally decreases with a decrease in T. In other words, the search through the solution space is more likely to jump to a worse solution at the beginning of the search as opposed to towards the end. The ability to jump to a solution of lower quality effectively minimizes the tendency to get stuck at a local optimum.

Applicability

Simulated annealing is often used when the search space is discrete (e.g., all tours that visit a given set of cities), whereas BECDLMST is a problem with a continuous solution space. Moreover, the concept of solution neighborhood is hard to define in BECDLMST. For a given tree structure, the position of each rendezvous point is

a variable that can be changed on the continuous 2D plane. Moreover, changes in tree structure are hard to categorize under the neighborhood relationship, mainly because simple changes would not usually lead to valid solutions under the given constraints. As a result, simulated annealing is not chosen for the problem at hand.

5.1.2 Genetic Algorithms

Genetic algorithms (GAs) are search algorithms based on the mechanics on natural selection and natural genetics [38, 39]. In GAs, a population of strings, which encode candidate solutions (called individuals) to an optimization problem, are evolved towards better solutions. A fitness function is used determine the quality of a candidate solution. The algorithm begins with a population of randomly generated candidate solutions following which evolution begins. A simple genetic algorithm that yields good results in many practical problems is composed of three operators are applied at each iteration (generation) [38] :

- *Selection*: The *selection* operator determines those individuals in the population that survive to participate in the production of the next population. Selection is based on the value of the fitness function, or the fitness of individual members of the population, such that members with greater fitness levels tend to survive.

- *Crossover*: Crossover recombines traits of the selected individuals in the hope of producing a child with better fitness levels than its parents. Crossover is accomplished by swapping parts of strings representing two individuals in the population.

- *Mutation*: *mutation*, introduces some sort of modification in the population members and prevents the search of the space from becoming too narrow.

Evolution strategies topology natural evolution by asexual reproduction with mutation and selection.

Genetic algorithms are different from more normal optimization and search procedures in four ways :

- GAs work with a coding of the parameter set, not the parameters themselves,
- GAs search from a population of points, not a single point,
- GAs use payoff (objective function) information, not derivativess or other auxiliary knowledge,
- GAs use probabilistic transition rules, not deterministic rules.

Applicability

The problem with the use of GAs for BECDLMST is the need for encoding candidate solutions. Since rendezvous points exist in a continuous space, encoding a solution isn't straightforward. On the other hand, it would be possible to encode a tree structure. However, even in the latter case, any mutation or crossover applied would very likely yield an invalid solution that does not conform to the constraints. In other words, it would be quite impossible to have an encoding of the tree that covers all possibilities while ensuring all (or most) permutations of the string represent valid solutions. Moreover, if an encoding simply represented the tree structure alone, it would be computationally expensive to evaluate its fitness (given by the best possible rendezvous point locations for said structure). As a result, we choose not to employ GA based methods.

5.1.3 Particle swarm optimization

The particle swarm optimization (PSO) algorithm was introduced by [40] and it is based on the social behavior of biological organisms that move in groups such as birds and fishes. It also has some ties to evolutionary algorithms such as GA and was originally developed to solve nonlinear optimization problems.

The basic element of PSO is a particle, which can fly throughout the search space towards an optimum by using its own information as well as that provided by other particles comprising its neighborhood. In PSO, a particle's neighborhood is the subset of particles which it is able to communicate with. Depending on how the neighborhood is determined, the PSO algorithm may embody the *gbest* topology, where each particle is connected to every other particle in the swarm so that it can obtain information from them. In other words, the neighborhood of a particle is the entire swarm. Alternatively, in the *lbest* topology a particle is not able to communicate with all other particles but only with some of them. The most simple *lbest* topology, also known as *ring* topology, connects each particle to only two other particles in the swarm, usually the nearest ones.

Each particle is a D dimensional vector X_i and has a velocity V_i. Some objective function f (fitness) is given.
P_i : so far best position of particle i
$P_g = \max_i\{P_i\}$

Evolution equation:
$V_i(k+1) = wV_i(k) + c_1r_1(P_i - X_i(k)) + c_2r_2(P_g - X_i(k))$
$X_i(k+1) = X_i(k) + V_i(k+1)$
with $w \in [0,1], c_1 > 0, c_2 > 0$ being constants and r_1, r_2 being random numbers in $[0,1]$. Iteration terminates after some time or after $f(P_g)$ reaches some value.

Applicability

The advantage of PSO is that the classical version was designed to operate in a continuous search space. Given that BECDLMST involves rendezvous points located on a continuous 2D plane, we choose to employ ideas from PSO to solve the problem at hand. We know that a solution to BECDLMST comprises of the entire tree structure along with positions for the rendezvous points. As such if PSO were to be applied directly, one particle would represent an entire candidate solution, whose dimensionality would be fairly huge. Instead, we choose to represent every rendezvous point as a particle capable of moving on the 2D plane. The complete set of particles along with connections between them would then represent a candidate solution. Particles interact with each other and cause each other to move at certain velocities dictated by an appropriate policy. Particles also perform the additional function of connecting/disconnecting from other particles to restructure the tree. The solution therefore evolves with each iteration to lead to a near optimal BECDLMST. The algorithm is presented in detail in the following section.

5.2 Evolutionary PSO-like algorithm

This section describes the proposed anytime heuristic to find a near optimal solution to BECDLMST. The algorithm causes rendezvous points to demonstrate a self-organizing behavior much like in Particle Swarm Optimization. The idea we use is to start with a Steiner tree that meets the agent count restriction, i.e., M edges on the tree. The positions of the RPs are then iteratively updated to lead to a lower latency network. Restructuring of the network is also performed by adding/removing RPs and re-attaching vertices in the tree. All updates performed within an iteration always ensure that the tree provides exactly M agent paths

connecting all N ground nodes at the end of each iteration, thus making it an anytime algorithm. Definitions:

$$\begin{aligned} \mathbb{G} \subset \mathbb{R}^2 &:= \text{set of ground nodes} \\ g_i \in \mathbb{G} &:= i^{\text{th}}\text{ground node} \\ \mathbb{V} \subset \mathbb{R}^2 &:= \text{set of RPs} \end{aligned}$$

5.2.1 Starting configuration

Let the Minimum Enclosing Circle (MEC) for $\mathbb{G}$ be centered at c_M and have a radius of r_M. We use c_M as root to construct the MDST as it minimizes the diameter of the tree and serves as a good starting point. We start with the least possible number of RPs for a given N, M combination. Edge-count is bounded by M and is always 1 less than number of vertices in the tree. Each leaf vertex can be one or more ground nodes (if more than one, the edge connecting this vertex to the tree is the path from the parent RP through the group of ground nodes). The notion of multiple ground nodes in one leaf vertex can be better understood by observing Figure 4.7(c) which shows branches with multiple ground nodes serviced by the same agent. All ground nodes on one such branch would be part of the same leaf vertex. As a result, the bound on number of RPs is given by:

$$\max(1, M - (N - 1)) \leq |\mathbb{V}| \leq M - 1 \tag{5.1}$$

Note that we do not consider the trivial case of $M = 1$ for which the solution is the TSP tour. So we always introduce at least 1 RP. We now have 2 classes of N, M configurations that need to be handled differently.

1. $M \geq N$: In this case, for least number of RPs, every ground node is treated as a single vertex. The first RP is introduced at c_M. All remaining RPs

are distributed proportionately along the edges connecting the c_M to each ground node. More formally, let $\mu(g_i) = \frac{\|g_i - c_M\|}{\sum_i \|g_i - c_M\|}(M - N)$ represent g_i's share of RPs. If *sortedNodes* is the list of ground nodes sorted in descending order by $(\mu(g_i) - \lfloor \mu(g_i) \rfloor)$, the number of RPs between c_M and g_i is given by $\lfloor \mu(g_i) \rfloor + 1$ for the first $M - N - \sum_i \lfloor \mu(g_i) \rfloor$ nodes in *sortedNodes*, and $\lfloor \mu(g_i) \rfloor$ for the remaining.

2. $M < N$: Only 1 RP can be inserted in this case and is placed at c_M. The ground nodes however need to be partitioned into M groups (one for each agent), such that the longest agent path is minimized. The smaller the M, the more the problem turns into a TSP problem. For the sake of speed, we choose to use a heuristic that works reasonably well for bigger $\frac{M}{N}$ ratios. The MEC is divided into M sectors, each with either $\lfloor \frac{N}{M} \rfloor$ or $\lfloor \frac{N}{M} \rfloor - 1$ ground nodes, such that every ground node belongs to exactly one sector. For each sector, a path is then drawn from c_M through the ground nodes in increasing order of distance from c_M.

Although case dependent, the average maximum latency in the starting configuration can be estimated at $(2 + 2\frac{N}{M})\frac{r_M}{v_{max}}$.

5.2.2 Iterative Tree Evolution

The initial tree that was built is evolved iteratively to lower the network latency. Each iteration performs a series of updates to the structure of the tree by moving/introducing/deleting RPs such that at the end of each iteration the tree still provides exactly M agent paths connecting all N ground nodes. The algorithms for each kind of update performed are laid out in Algorithms 2 - 6. Here we describe the rationale behind each step and its effects. 3 kinds of updates are performed in each iteration:

Modify positions of vertices without affecting structure of tree

Algorithm 2 RP Position Update

for each $v \in \mathbb{V}$ **do**
  $f_v :=$ furthest neighbor of v
  $s_v :=$ second furthest neighbor of v
  Starting η at 0.1 and reducing it exponentially, compute
  $v_{new} \leftarrow v + \eta(f_v - v + s_v - v)$
  until $\|f_{v_{new}} - v_{new}\| \leq \|f_v - v\|$ becomes true
  $v \leftarrow v_{new}$
end for

This category refers to the first step in the iteration - RP position update - as detailed in Algorithm 2. In this step, every RP is moved in order to minimize the maximum distance from any neighbor (i.e. any child or parent). For each RP, its two furthest neighbors apply an attractive force on the RP, proportional to the length of edges connecting them to the RP. The effect of the forces is dampened by a factor of η such that distance to the furthest neighbor does not increase after the update. For a given tree structure, the RP positions always converge to a stable state such that $\eta \approx 0$ for all RP position updates in an iteration. When this happens, we consider the structure to have *settled*, giving the lowest possible λ_h for that structure. It is only on *settling*, that the structure of the tree is modified with the following steps.

Modify structure of tree without increasing n_h or λ_h

This category includes the next 2 steps in the iteration - Re-attaching ground nodes and RPs - as detailed in Algorithms 3 and 4. Each leaf vertex (could be one or more ground nodes) is re-attached to its closest RP. If the leaf vertex is the only child of its parent, it tries to attach itself to any other RP (not necessarily closest) without increasing λ_h so that the parent RP is freed. For each ground node, g, we

Algorithm 3 Re-attach Ground Nodes

for each $g \in \mathbb{G}$ directly connected to some $v_1 \in \mathbb{V}$ **do**
  if g is only child of v_1 **then**
    Doing a breadth-first traversal of the tree, find the first $v_2 \in \mathbb{V}$ that satisfies $\|v_2 - g\| + rlen(g) \leq \lambda_h$
  else
    $v_2 \leftarrow \underset{v \in \mathbb{V}}{\text{argmin}} \|v - g\|$
  end if
  if $v_2 \neq v_1$ **then**
    Detach g from v_1 and attach as child to v_2
  end if
end for

Algorithm 4 Re-attach Rendezvous Points

for each $v \in \mathbb{V}$ **do**
  Doing a breadth-first traversal of the tree, find the first $v_2 \in \mathbb{V}$ such that $height(v_2) > height(v) \wedge \|v_2 - v\| < \|\text{parent}(v) - v\|$
  if valid v_2 is found **then**
    Detach v from parent(v) and attach as child to v_2
  end if
end for

define two partial lengths of the edge through g, that are used in the algorithm:

$$\begin{aligned} rlen(g) &:= \text{partial edge length from } g \text{ to outermost} \\ &\quad\ \ \text{ground node} \\ plen(g) &:= \text{partial edge length from } g \text{ to parent RP} \end{aligned}$$

On the other hand, each RP is re-attached to any other closer, yet viable parent. An RP, v_2, is deemed a viable parent of v_1, if $height(v_2) > height(v_1)$, where $height(v)$ is defined as the height of the subtree rooted at v. Neither of the re-attaching steps increases n_h or λ_h. The new structure however, could potentially reduce λ_h through RP position updates.

Modify structure of tree increasing/decreasing number of RPs, possibly affecting n_h and λ_h

Algorithm 5 Group Ground Nodes

for each $g_1 \in \mathbb{G}$ directly connected to some $v \in \mathbb{V}$ **do**
  Find closest $g_2 \in \mathbb{G}$ with $rlen(g_2) = 0$ such that $rlen(g1) + plen(g_2) + \|g_1 - g_2\| \leq \lambda_h$
  if valid g_2 found **then**
    Detach g_1 from v and attach to g_2
    Insert new RP in the middle of longest edge in tree
    Break from loop
  end if
end for

Algorithm 6 Ungroup Ground Nodes

for each $g \in \mathbb{G}$ with $rlen(g) = 0$ and not directly connected to an RP **do**
  if $plen(g) = \lambda_h$ **then**
    $newParent \leftarrow \underset{v \in \mathbb{V}}{\text{argmin}} \|v - g\|$
    Detach g from parent(g) and attach as child to $newParent$
    $vRem \leftarrow \underset{v \in children(v_{root})}{\text{argmin}} \|v - v_{root}\|$
    Detach all $children(vRem)$ from $vRem$ and attach as children to v_{root}. Detach $vRem$ from v_{root}, delete $vRem$, and break from loop
  end if
end for

This category includes the last 2 updates in the iteration - Grouping and ungrouping of ground nodes - as detailed in Algorithms 5 and 6. Grouping is done by considering each leaf vertex and checking if it can be grouped with, and attached to the end of another leaf vertex without increasing λ_h. We know that grouping ground nodes into a leaf vertex has the effect of reducing the number of edges in the tree. For every edge that is removed by grouping, a new edge has to be introduced so that the total number of edges remains constant. A new edge is therefore introduced by adding an RP right in the middle of the longest edge in the tree (λ_h length) with preference to the one closest to the root (i.e. the closest bottleneck edge to the root).

Ungrouping is done only when no more grouping is possible. We look for an edge, e, that passes through more than one ground node, and has a length of λ_h. Considering e to be the network's bottleneck, we remove the outermost ground node

on e from its group and connect it to its closest RP, thus increasing the number of edges. In order to bring back the number of edges to M, the closest RP to the root, $vRem$, is deleted and all of $vRem$'s children are attached to the root.

At the end of each iteration, we check if any childless RP exists due to grouping and move it to the middle of the longest edge in the network.

Now, increasing the number of RPs, could increase n_h without decreasing λ_h, while decreasing the number of RPs, could increase λ_h without decreasing n_h. For this reason, only one change is performed in this step for any iteration and subsequently, the network is allowed to *settle*. The first change that caused an increase in network latency is recorded and the state of the tree before the change is saved. After $N + M$ consecutive changes that fail to lower network latency below that of the saved state, the saved tree is restored and the first of the $N + M$ changes is never performed again.

The algorithm comes to an end when the same tree is restored $N + M$ times, or if absolutely no change occurs in an iteration. The sequence of steps within one iteration are laid out in Algorithm 7.

Algorithm 7 Steps in an iteration

```
RP Position Update
if η ≈ 0 for all position updates, i.e. settled then
   Re-attach Ground Nodes
   Re-attach Rendezvous Points
   Group Ground Nodes
   Ungroup Ground Nodes
   Move childless RP to middle of longest edge in tree
end if
```

Figure 5.1 shows the algorithm running on one ground node configuration instance. The example shown uses $N = 10, M = 10$. Using κ to represent the number of iterations, three snapshots of the tree at different stages of evolution are shown:

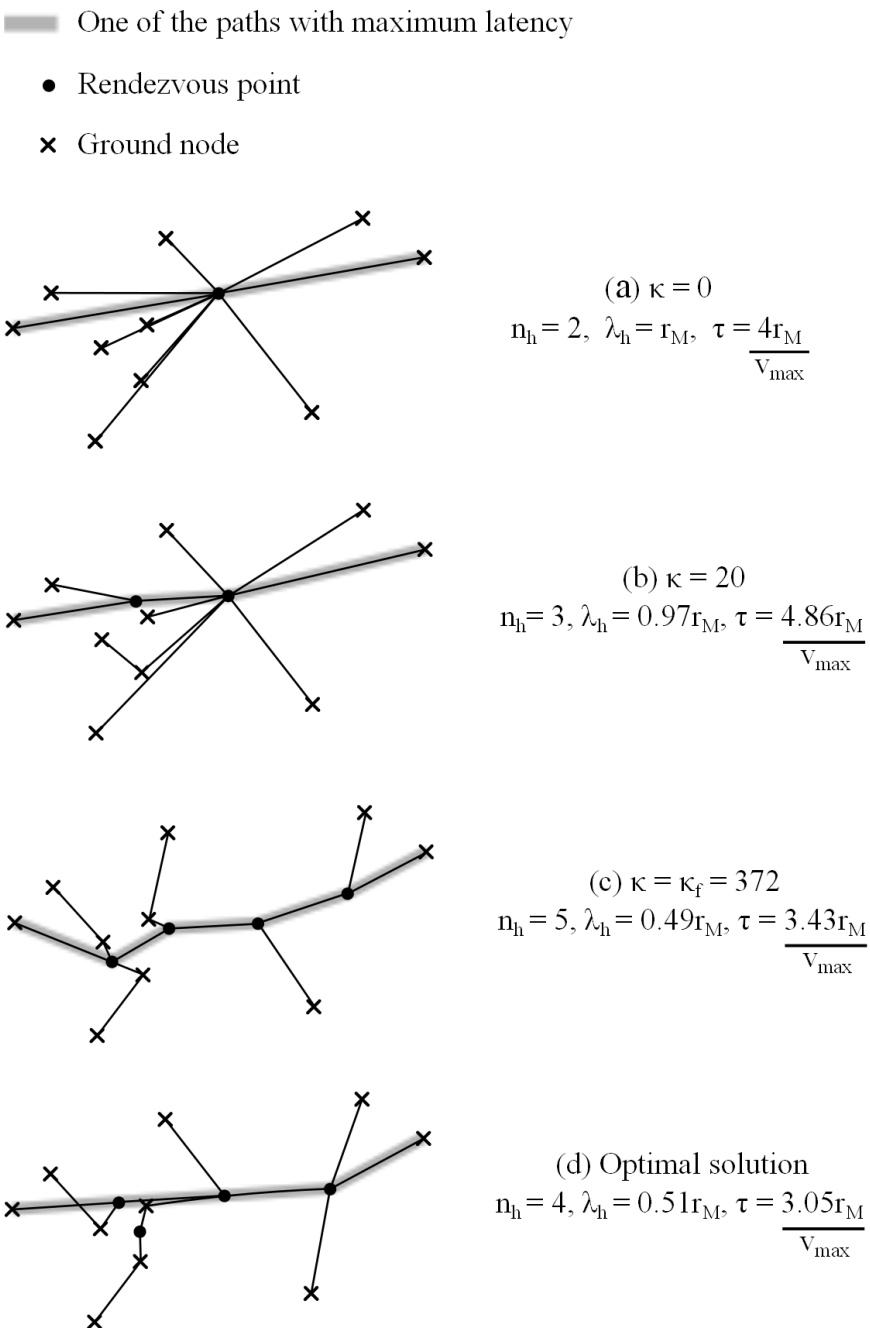


Figure 5.1: (a,b,c) Tree at different stages in the proposed algorithm for a sample ground node configuration ($N = 10, M = 10$) (d) Optimal BECDLMST generated by exhaustive exponential algorithm (Chapter 4) for same ground node configuration

(a) Starting configuration at $\kappa = 0$ iterations.

(b) An intermediate stage ($\kappa = 20$) where re-attaching and grouping of ground nodes has taken place. As discussed earlier, steps that change the structure have the potential to increase network latency. Whether the increase is temporary will be known on *settling*.

(c) Final tree structure at algorithm termination ($\kappa = 372$)

Figure 5.1(d) shows the optimal BECDLMST for the exact same ground node configuration computed using the exact exponential algorithm from Chapter 4.

5.3 Results

Table 5.1: Experimental results using heuristic for small N, M with comparison against optimal BECDLMST

	M=3			M=5		
N	Normalized τ_f	$\frac{\tau_f}{\tau_{opt}}$	$\kappa_{1.05\tau_f}$	Normalized τ_f	$\frac{\tau_f}{\tau_{opt}}$	$\kappa_{1.05\tau_f}$
3	4.00±0.00	1.00±0.00	1±0	3.23±0.34	1.00±0.00	13±10
5	5.90±1.35	1.02±0.03	17±12	3.85±0.25	1.05±0.10	10±17
10	9.37±2.26	1.06±0.10	13±11	4.98±0.83	1.07±0.03	45±44
20	14.70±3.81	1.11±0.15	20±15	7.84±1.20	1.09±0.11	17±19

	M=10			M=20		
N	Normalized τ_f	$\frac{\tau_f}{\tau_{opt}}$	$\kappa_{1.05\tau_f}$	Normalized τ_f	$\frac{\tau_f}{\tau_{opt}}$	$\kappa_{1.05\tau_f}$
3	2.60±0.15	1.00±0.00	55±33	2.32±0.08	1.00±0.00	159±122
5	2.87±0.15	1.02±0.02	26±28	2.40±0.08	1.01±0.02	120±73
10	3.58±0.29	1.04±0.08	156±248	2.73±0.14	1.01±0.02	87±69
20	4.27±0.38	1.05±0.06	78±80	3.31±0.27	1.04±0.10	466±411

We run our algorithm on numerous ground node configurations to test the quality of solutions produced as compared to the optimal BECDLMST as well as observe the number of iterations taken to produce acceptable results. For every value of N, we generate 50 different random ground node configurations. For every configuration, we apply our algorithm with different values of M. So for each N, M

combination we have 50 experiments. In each experiment, we record 2 pieces of information: 1) τ_f, final network latency when the algorithm terminates, and 2) $\kappa_{1.05\tau_f}$, number of iterations taken to reach a network latency 5% away from τ_f. The number of iterations for the algorithm to terminate is usually much larger than $\kappa_{1.05\tau_f}$. However, the benefit of an anytime algorithm allows us to stop earlier for practical purposes. $\kappa_{1.05\tau_f}$ represents the number of iterations at which the algorithm could have been stopped to achieve a network latency very close to the final achievable network latency. The optimal BECDLMST is then found using the exact exponential algorithm laid out earlier in Chapter 4. As described, the exact exponential algorithm enumerates all possible rendezvous points for any given ground node configuration and finds the combination with minimum network latency. The network latency achieved by the optimal BECDLMST, τ_{opt}, is compared against τ_f, and the ratio $\frac{\tau_f}{\tau_{opt}}$ is recorded. $\frac{\tau_f}{\tau_{opt}}$ signifies the offset of the tree generated by the heuristic from the optimal BECDLMST. Since finding the optimal BECDLMST takes exponential time, the data set sizes for which comparison is done are kept small. We run N and M through four different values of 3, 5, 10 and 20, the results for which are shown in Table 5.1. The values in the table show the mean and standard deviation over 50 runs for each N, M setting. Note that τ_f is normalized and presented as a co-efficient of $\frac{r_M}{v_{max}}$.

In order to assess the scalability and robustness of the algorithm, we run the same experiments with much larger data sets with a wider range of $\frac{N}{M}$ ratios. N is varied from 50 to 10000 and M is varied from 50 to 1000. For these data sets however, comparison with the optimal solution is infeasible. The results obtained are presented in Table 5.2.

We observe that network latency consistently increases with increase in N and decreases with increase in M. Additionally, for the same $\frac{N}{M}$ ratio, network latency reduces with an increase in N or M, which suggests that M has a stronger effect on

Table 5.2: Experimental results for large N, M

	M=50		M=100		M=1000	
N	Normalized τ_f	$\kappa_{1.05\tau_f}$	Normalized τ_f	$\kappa_{1.05\tau_f}$	Normalized τ_f	$\kappa_{1.05\tau_f}$
50	3.25±0.13	3047±1163	3.06±0.05	1544±884	2.09±0.01	52±69
100	3.90±0.07	637±105	3.20±0.18	3515±977	2.17±0.01	842±921
1000	5.46±0.09	812±241	4.17±0.02	2311±1809	3.24±0.08	4390±657
10000	23.73±0.24	150±42	8.05±0.11	714±722	4.00±0.00	5199±3942

network latency as compared to N. As for proximity to optimality, our algorithm performs very well when the $\frac{N}{M}$ ratio is small. The reason for this is that for large $\frac{N}{M}$ ratios, the number of rendezvous points in the tree is small, thus making the problem more of a TSP problem than a rendezvous point placement problem. Since grouping of ground nodes is done in a greedy fashion in our algorithm (for the sake of speed), the quality of the solution obtained for large $\frac{N}{M}$ ratios is limited. However, in general, for the data sizes tested, our algorithm produced solutions with network latencies within 15% in excess of τ_{opt}, which is a small loss in solution quality for a significant improvement in computation time.

From the algorithm, we know each iteration on its own has $O(N^2 + M^2)$ time complexity. The number of such iterations required to achieve convergence, represented by $\kappa_{1.05\tau_f}$, generally increases with an increase in N or M. However, the highest values occur as the $\frac{N}{M}$ ratio gets closer to 1. While running our experiments, we observed that the number of feasible structural changes to the tree was higher for $\frac{N}{M}$ ratios close to 1, thus leading to more number of iterations. For small values of $\frac{N}{M}$, ground nodes did not have to be grouped, and most were already connected to their nearest RP. The tree structure stabilized fairly quickly. For large values of $\frac{N}{M}$, each group of ground nodes was huge, and any change in structure usually resulted in an increase in λ_h. As a result, the number of feasible changes was low. The end result is that the algorithm is capable of producing near optimal results on completion. Convergence to a latency close to τ_f happens much earlier and for

practical purposes, the anytime nature of the algorithm allows early termination.

Although tables 5.1 and 5.2 present algorithm runtimes using the system independent metric - number of iterations, actual time data can sometimes prove to be useful. To provide a ballpark, on our 2.3GHz Core 2 Duo system, the $(N = 20, M = 20)$ case took 10s on average to complete all iterations and less than 1s to reach $1.05\tau_f$. The $(N = 100, M = 100)$ case on average took about 500s for all iterations and 70s to reach $1.05\tau_f$.

Table 5.3: Comparison of normalized τ (network latency), as produced by our heuristic and FRA [23] for small N, M values

	M=3			M=5		
N	Normalized τ_{FRA}	Normalized τ_f	Percent improvement	Normalized τ_{FRA}	Normalized τ_f	Percent improvement
3	17.05±5.06	4.00±0.00	76.53	-	3.23±0.34	-
5	13.71±2.77	5.90±1.35	56.96	13.38±2.79	3.85±0.25	71.22
10	13.51±1.53	9.37±2.26	30.64	12.01±1.59	4.98±0.83	58.53
20	15.97±1.50	14.70±3.81	7.95	13.60±1.50	7.84±1.20	42.35
	M=10			M=20		
N	Normalized τ_{FRA}	Normalized τ_f	Percent improvement	Normalized τ_{FRA}	Normalized τ_f	Percent improvement
3	-	2.60±0.15	-	-	2.32±0.08	-
5	-	2.87±0.15	-	-	2.40±0.08	-
10	11.51±1.93	3.58±0.29	68.90	-	2.73±0.14	-
20	10.93±1.66	4.27±0.38	60.93	10.35±1.74	3.31±0.27	68.01

In addition to a comparison against the optimal BECDLMST, the heuristic presented in this chapter is also compared against FRA. Table 5.3 presents comparison of results for small values of N, M ranging between 3 and 20. Table 5.4 on the other hand presents a comparison for very large values of N, M ranging up to 10000 nodes and 1000 agents. We observe that the network latencies achieved by the approximation to BECDLMST, still outperform FRA considerably. In particular, we notice that for high values of M, our heuristic achieves significantly lower network latencies regardless of number of ground nodes. We believe the reason is that with a high number of agents, there is far greater opportunity to strategically place rendezvous points, which is the advantage that BECDLMST has over FRA.

Table 5.4: Comparison of normalized τ (network latency), as produced by our heuristic and FRA [23] for large N, M values

	M=50			M=100		
N	Normalized τ_{FRA}	Normalized τ_f	Percent improvement	Normalized τ_{FRA}	Normalized τ_f	Percent improvement
50	9.77±1.36	3.25±0.13	66.73	-	3.06±0.05	-
100	9.62±1.16	3.90±0.07	59.46	9.63±1.18	3.20±0.18	66.77
1000	19.87±1.61	5.46±0.09	72.52	14.86±0.96	4.17±0.02	71.94
10000	92.25±1.79	23.73±0.24	74.27	54.67±2.56	8.05±0.11	85.27

	M=1000		
N	Normalized τ_{FRA}	Normalized τ_f	Percent improvement
50	-	2.09±0.01	-
100	-	2.17±0.01	-
1000	9.71±0.56	3.24±0.08	68.90
10000	14.98±0.69	4.00±0.00	73.30

5.4 Summary and Contributions

An anytime heuristic using update rules and tree evolution strategies was presented that enabled the self-organization of rendezvous points to generate close approximations of the BECDLMST. The algorithm was designed such that at the end of every iteration, the tree satisfied the edge and rendezvous point constraints, thus giving a valid solution and allowing an anytime termination. Experiments showed that the algorithm was capable of handling very large data sets in terms of ground nodes and agents. In general, the algorithm performed better with more agents available.

The main contribution of this chapter is the utilization of ideas from PSO to serve in closely approximating the BECDLMST. The heuristic brings the computation complexity from an exponential one down to a quadratic one. Moreover, the heuristic is designed to be anytime in nature, so that early termination is an option. In all, the ideas presented in this chapter, make BECDLMST a feasible solution to the problem described in Chapter 2.

Chapter 6

Problem Expansion under realistic conditions

The first half of the thesis talked about the agent path planning problem under the context of complete information. The locations of ground nodes were known beforehand and ground nodes were assumed to be stationary. However, when realism is brought into the picture, a number of assumptions that were laid out in the original problem definition (Chapter 2) cease to hold.

In a realistic situation, agents might not know the locations of ground nodes beforehand. Ground nodes will need to be searched for under such circumstances. While this can be done by listening for wireless signals, ground nodes cannot be expected to transmit signals continuously. The reason is that ground nodes, who can potentially be survivors or rescuers, can turn wireless off and on anytime for saving battery or any other purpose. A wireless signal that is continuously on, cannot be assumed of the ground nodes. In other words, ground nodes can potentially be intermittent. What were originally considered to be nameless agents in the theoretical sense, will now have to be UAVs mounted with wireless equipment.

This introduces the problem of autonomous control of the UAVs, with the additional challenge of accurate navigation under harsh weather conditions. Wireless equipment carried by these UAVs would have realistic communication limitations in terms of range, packet loss, etc. The challenges posed by the above mentioned factors can be listed out as follows.

6.1 Challenges

The key challenges that need to be addressed in trying to solve this problem are:

1. Lack of global information in agent planning : Each UAV has a limited communication range. As a result, the only information that a given agent can have is its own local information and information obtained from neighboring agents. Action decisions at each agent have to be made based on this partial information.

2. Intermittent and mobile ground nodes : In order to incorporate survivors, who could be on the move and not necessarily have their WiFi turned on at all times, we model these ground nodes as intermittent and mobile radio sources. As a result, the coverage problem gets extended to become a persistent search problem.

3. Opposing trade-offs in a dynamic environment : Visit frequency provides a measure for search quality while packet latency provides a measure for tethering quality. When one is emphasized, the other suffers. Moreover, the environment, which constitutes the position of ground nodes, number of UAVs on the field, etc., is dynamic in nature. The challenge then is to balance the trade-offs and at the same time, adapt with changes in the environment.

4. Limited communication bandwidth : Information exchange for coordination has to take place over wireless links between UAVs. The bandwidth of such links is limited and is to be shared with data communication between ground nodes. Therefore, there is a need to minimize the amount of information exchanged between agents.

5. Adverse wind effects : UAV control should be capable of handling wind effects that often lead to path drift. The problem here is that small UAVs are much more easily affected by winds and the challenge is overcome these effects to provide precise navigation.

6.2 Problem Redefinition

To incorporate the above challenges, we redefine the problem as a control and coordination problem requiring a decentralized solution where every UAV thinks and acts based on its own and locally exchanged information.

Let us assume an $X \times Y$ grid representing the disaster struck area. A set of K agents (UAVs) are dispatched to start operations from random positions in the grid. To account for the worst case scenario, agents are assumed not to have *a priori* information about positions of rescue teams and survivors. The agents however have knowledge of the base node's position, which is along one of the edges of the grid, bordering the disaster struck area. All ground nodes are capable of movement, and are assumed to remain within the boundaries of the grid (which can be chosen to be large enough to ensure the validity of this statement). Ground nodes are assumed to have a maximum speed of $10ms^{-1}$ and are not expected to transmit signals all the time. In other words, a given ground node could be an intermittent signal emitter. The task of the agents then is to constantly keep

searching for ground nodes, and establish communications between those that have already been found. We shall use the term *tethering* to refer to the task of enabling communications for ground nodes via UAVs. The challenge is to maximize both search frequency (of the entire area) as well as bandwidth available to each ground node while minimizing the latency for packet delivery. The bandwidth available to a ground node can be considered to be directly proportional to the amount of time it has an agent within communication range, also known as service duration. Since every ground node has only one transmitting radio and since the data rate is capped by the constant wireless link capacity, the only variable that affects the amount of data transmitted per unit time is service duration. As a result, the aim is to maximize quality, Q, of the search and relay operation,

$$Q = \frac{f_{avg} s_{avg}}{\tau} \tag{6.1}$$

where f_{avg} = average visit frequency for all grid cells measured over the last T_W time units

$$= \frac{1}{XY} \sum_{i=1}^{X} \sum_{j=1}^{Y} f_{i,j}$$

s_{avg} = average service time over the last T_W time units for ground nodes

$$= \frac{1}{|\mathbb{G}|} \sum_{i \in \mathbb{G}} s_i$$

τ = network latency (more accurately maximum latency for all packets delivered in the last T_W time units)

$$= \max_{i=1}^{P} \tau_i$$

Now, the problem of maximizing Q is already a complex one owing to the tradeoff between visit frequency and packet latency as well as the intermittent and mobile nature of ground nodes. The combined problem of coverage, search and tethering

is a novel one. When subject to real life situations of wind and communication constraints, the problem gets even harder, especially in the control domain. Precise navigation of each aircraft despite wind conditions involving crosswinds, gusts and turbulence, becomes an important issue when trying to coordinate multiple UAVs. If UAVs get displaced from their intended paths, the coordination algorithm will also suffer. As a result, the control component has the additional problem of adapting to wind effects, specifically lateral displacement caused by crosswinds. Communication restrictions on the other hand make it impractical to use a centralized solution. As a result the problem requires a decentralized solution wherein each agent makes decisions based on local and partial information.

In summary, the aim is to devise a complete solution to the above described combined problem of coverage, search and tethering, incorporating both aspects of control and coordination under realistic communications and wind conditions.

Chapter 7

Control and Coordination architecture

In this chapter we explore the possible architectures for designing the control and coordination system to be deployed on UAVs. We look at related work in this area and subsequently present our proposed architecture. The aim of this chapter is to present a bird's eye view of the system that is described in detail in Chapters 8 and 9.

7.1 Related work

In the multiagent control and coordination literature, although there have been works on the individual tasks of coverage, search and tethering there has been no work that considers the combined problem of all three tasks, let alone in realistic conditions with winds and communication limitations. In the control domain, precise waypoint navigation is receiving increased attention owing to the increase in lightweight UAV applications. Recent work in the controls area is discussed in

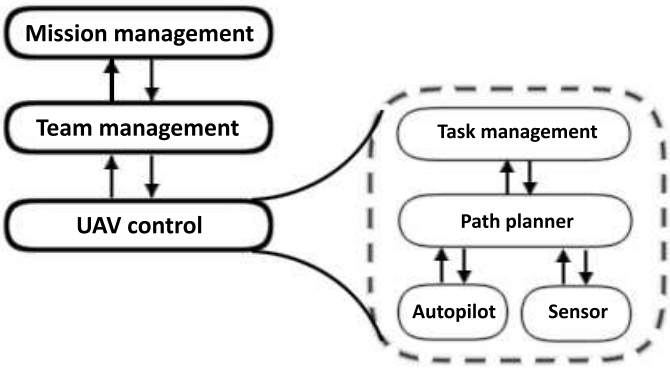


Figure 7.1: UAV team control & coordination architecture

more detail in Section 8.2. In the coordination domain, although the problem of coverage and search has been studied in the multiagent literature, little has been done in the area of tethering. Existing works catering to each of the individual tasks are discussed in more detail in Section 9.1. As for the overall architecture, there have been ideas proposed for other types of UAV-based control and coordination tasks. We shall take a look at some of the most relevant ones here.

Numerous works [41, 42, 43] use a hierarchical control and coordination architecture which is generally represented as shown in Figure 7.1. The mission management layer contains the mission objectives which could possibly be specified by a human operator. At the team management layer, each team is given one or more objectives. These objectives are then decomposed into a sequence of tasks, which are then assigned to individual vehicles. The team management layer is responsible for allocating, scheduling and possibly redistributing resources for each task. Having received the task(s), the UAV control layer has 3 hierarchical layers within itself to achieve the task(s). Task management executes the task(s) assigned to the UAV. The path planner layer is given the job of navigating the UAV in the operational environment in an optimal manner, avoiding obstacles and minimizing threat risks. At the lowest level, the autopilot takes care of flight control and

generally receives as input, path segments represented by GPS waypoints or yaw turn rate commands, from the path planner layer. The sensor management layer is responsible for handling issues specific to the sensors carried by the aircraft, such as onboard camera, etc. , depending on the nature of the mission.

The above architecture is a generalized version where each layer down until the path planner could be centralized or decentralized. In the centralized version [44, 45, 46, 47], all the state information from the distributed vehicles or agents is sent to a centralized agent, where it is operated on by a large decision and control program. The resultant individual plans and assignments are disseminated to the respective vehicles to be executed. Although centralized planning gives the advantage of complete information and hence more likely optimal control, this approach can lack significant robustness, is computationally complex, and does not scale well [48]. Realistic limitations on communication tend to make such centralized architectures impractical.

Decentralized versions [49, 50, 51, 52] using techniques such as distributed constraint satisfaction and stochastic dynamic programming have been proposed to overcome such limitations. A more detailed control and coordination architecture for the decentralized case as presented by Cole et al. [53] is shown in Figure 7.2. Here, the cooperative control and path planning module is equivalent to the mission management, team management, task management and path planning layers combined. The information exchanged between neighboring agents combined with the local state information is used to determine control commands.

Within these above architectures, the problem of wind effects can be solved at different layers. Although most works on UAV control and coordination ignore this issue, it is a pertinent one that has received recent attention owing to an increase in number of lightweight UAV swarm applications. A large number of

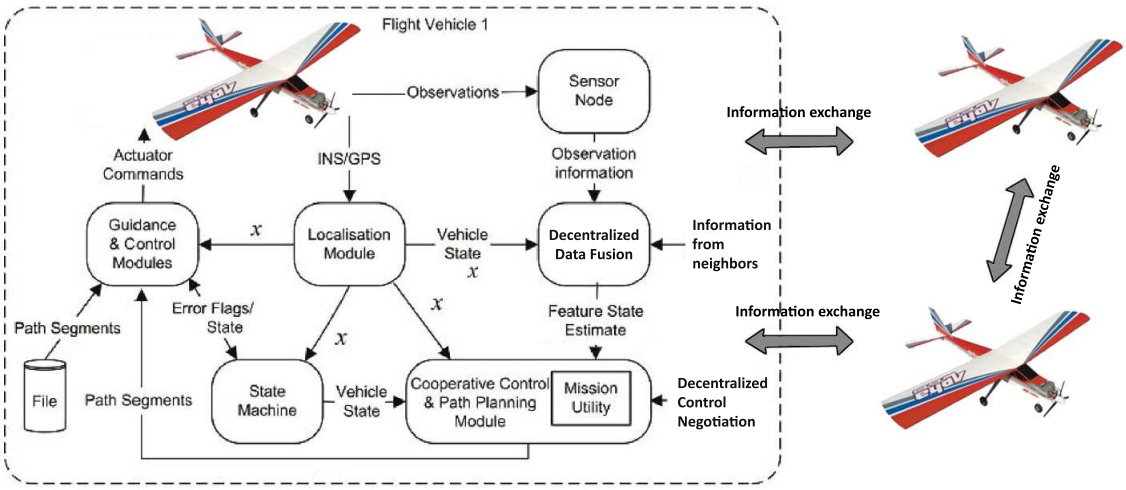


Figure 7.2: Decentralized control and coordination architecture (adapted from [53])

works address the issue of wind effects, by introducing a wind frame of reference [54, 55]. A conversion from the standard inertial frame to the "wind frame" is provided so that existing control algorithms that ignore winds will function in the presence of winds. However, the downside to this approach is the need for accurate wind speed and direction measurements. Due to the uncertain nature of winds and the inability to measure it accurately in flight, some works have opted to tackle the problem at the task management and path planner layers. For example, in [56], a "wind-robust" task assignment algorithm is designed to adapt task assignment and path planning to accommodate for wind uncertainty. Although this approach is reactive and does not require wind speed measurements, it becomes highly task dependent. The architecture becomes rigid in the sense that one layer cannot be easily modified without affecting the other. Any change in any of the layers will have to also account for wind effects.

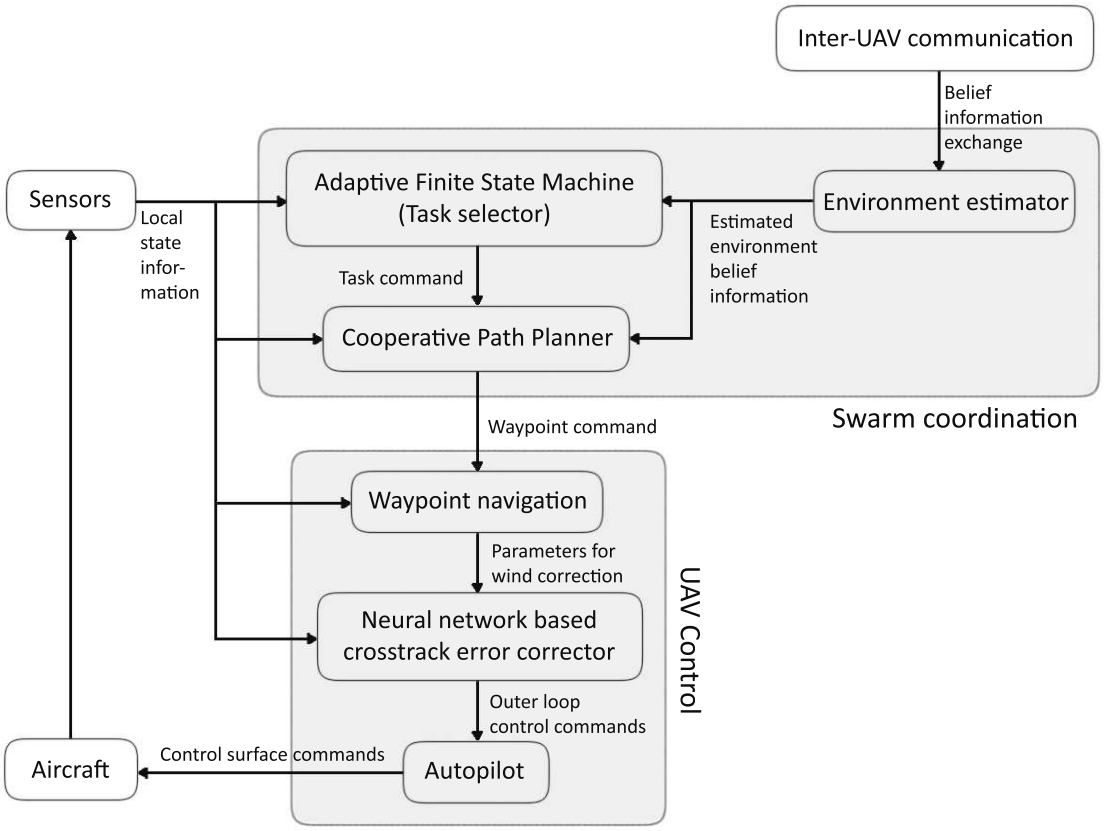


Figure 7.3: Proposed control and coordination architecture

7.2 Overall architecture

The architecture we use to tackle the expanded problem described in Chapter 6 is a decentralized hierarchical one very similar to that proposed in [53]. Figure 7.3 gives a clear idea of the proposed control and coordination architecture. From the larger perspective, it can be observed that the multiagent coordination component is built over the UAV control component. The coordination component assumes the job of producing an action (represented as a target waypoint). The action decision is based on current environment belief information and local state information. The local state information would include all data read by sensors onboard the aircraft such as GPS coordinates, heading, roll and pitch, ground speed, etc. The environment belief information on the other hand would include data about the entire search area obtained by fusing local belief information with that obtained

from neighboring agents. The adaptive finite state machine (FSM) is used to intelligently choose a task to perform. The task thus chosen would drive the cooperative path planner to decide on where the aircraft should head in order to achieve the commanded task. The reasoning done by the coordination component is thus reduced to a waypoint command value. A detailed explanation of the proposed coordination component is presented in Chapter 9.

By passing a waypoint command to the control component, the coordination component expects precise navigation to be achieved. Similar to how the higher layers did not have to compensate for wind effects in the "wind frame"-based solution, the coordination component can afford to ignore wind effects. In order for this assumption of precise navigation to be valid, the control component then needs to handle wind effects. But unlike the "wind frame"-based solution which requires hard-to-obtain, wind speed and direction measurements, we propose an intelligent, reactive controller using a neural network based cross-track error corrector. A detailed explanation of the proposed control component is presented in Chapter 8.

7.3 Summary

We presented the overall architecture of the system aimed at controlling individual UAVs and coordinating with other UAVs for the purpose of building a wireless backbone in a decentralized manner. We proposed a hierarchical structure where a coordination component performed communications and made higher level decisions to command a lower level control system.

Chapter 8

Robust UAV Control

In this chapter, we discuss the control algorithm for precise navigation of lightweight UAVs. In particular, we address the most prominent issue that affects control of lightweight UAVs, namely winds. We approach the problem by devising methods to handle the inherent nonlinearity of control and response of an aircraft in windy situations. In particular, we propose the use of neural networks that use cross-track error as the input parameter for making control decisions that minimize deviation from the intended flight path.

8.1 UAV Control basics

We first provide a quick look at the basics of aircraft control and introduce related terminology. Any aircraft has three axes of rotation, namely, the longitudinal axis, the vertical axis, and the lateral axis as shown in Figure 8.1. The axis that extends lengthwise (nose through tail) is called the longitudinal axis, and the rotation about this axis is called roll. Ailerons are used to control roll rate. They are attached to the wing and controlled in a manner that ensures one aileron will

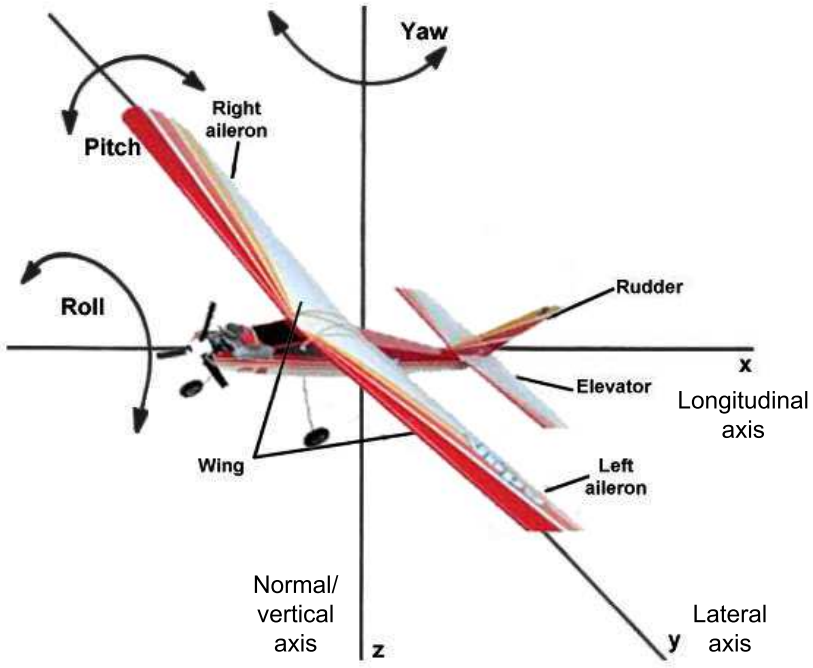


Figure 8.1: Axes of an aircraft

deflect downward when the other is deflected upward. When an aileron is not in perfect alignment with the total wing, it changes the wing's lift characteristics. To make a wing move upward, the aileron on that wing must move downward. A downward aileron produces more lift and an upward aileron upward reduces lift, thus causing the aircraft to roll.

The axis that extends crosswise (wing tip through wing tip) is called the lateral axis, and rotation about this axis is called pitch. Pitch is controlled using the elevator which is attached to the horizontal stabilizer (tail of aircraft). The elevator can be deflected up or down. With an upward elevator, the relative wind striking the top surface of the raised elevator pushes the tail downward. As the tail moves (pitches) downward, the nose moves (pitches) upward and the aircraft climbs. Similarly, a downward elevator causes the aircraft to pitch down and descend.

The axis that passes vertically through the center of gravity (when the aircraft is in level flight) is called the vertical axis, and rotation about this axis is called

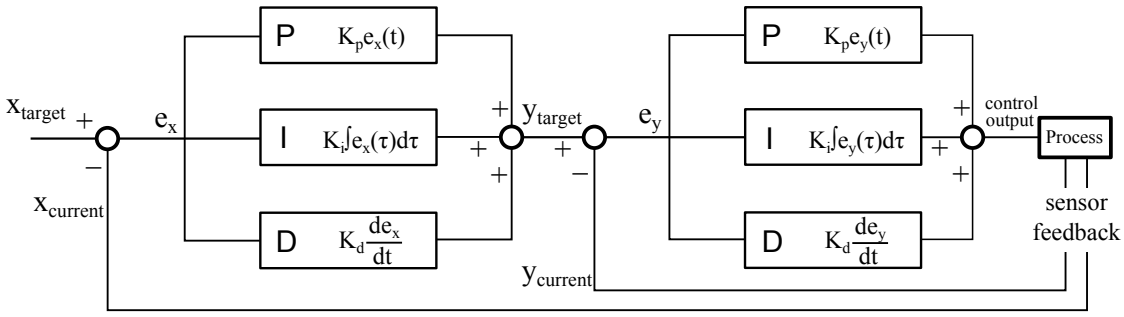


Figure 8.2: Dual PID Loop controller (Standard autopilot)

yaw. Yaw is controlled using the rudder. The rudder is a movable control surface attached to the vertical fin of the tail assembly. Considering the point of view from behind the aircraft, a leftwards rudder deflects the relative wind to the left, causing the tail to move to the right and the nose of the aircraft to yaw to the left. Similarly, a rightwards rudder causes the aircraft to yaw to the right.

The power control (throttle in piston-engined aircraft and electronic speed controller in motor-powered UAV) is used to change the amount of thrust from the engine. Adding power makes the aircraft speed up and reducing power makes the aircraft slow down. In essence, there are 3 actuators and the engine that can be used to control a UAV, the command values for which will need to be provided by the control system. The standard flight control system is a dual loop system as shown in Figure 8.2, where the inner loop is responsible for the fast dynamics associated with roll angle, pitch angle and yaw angle tracking. In other words, the inner loop is responsible to achieve a commanded angle in any of the 3 axes. The outer loop takes care of slower dynamics. The parameter controlled by the outer loop depends on the application the control system is built for. For the standard autopilot, it is associated with altitude hold, heading, etc. Both loops are built using proportional integral derivative (PID) controllers. A PID controller consists of a linear feedback control loop that takes a target system state and the current system state as input to produce a corresponding control output. Considering the first (i.e. outer) PID loop in Figure 8.2, e_x is the error in system state that is used

to generate three components - proportional, integral and derivative - which are then summed up to give the control output. In the dual loop system, output of the outer loop PID is treated as the target state for the inner loop PID.

8.2 Related work

The problem of precise waypoint navigation in the presence of winds has been a generally untouched area owing to the fact that large aircrafts are not severely affected by winds. As far as navigation towards a given waypoint is concerned, the de facto method has been to maintain aircraft heading towards the target waypoint. The heading hold is usually achieved through a proportional-integral-derivative (PID) controller [57, 58, 59]. In the case of lightweight UAVs (≈4kgs), the effects of wind become far more pronounced. The component of wind parallel to the flight path, known as head wind has the effect of decelerating or accelerating (and at the same time changing the altitude of) the aircraft depending on its direction. On the other hand, the wind component perpendicular to flight path, known as crosswinds, tend to deflect the flight path in the direction of the wind. As for lightweight UAVs, they are more likely to be deflected away from their intended flight path even by winds of moderate speeds (10 - 30kts). With the typical heading hold approach, such UAVs will most certainly miss the target waypoint as shown in Figure 8.3(a). Had the UAV taken a crab like approach in which the nose of the aircraft is pointed into the oncoming wind, the effective flight path could have been maintained so as to reach the target waypoint. The physics behind the crabbing approach for landing has been well studied [60] and an illustration of the way it works is shown in Figure 8.3(b).

A second drawback to many existing approaches is the linear nature of their controllers. For example, in [61], the feedback control provided by a PID controller

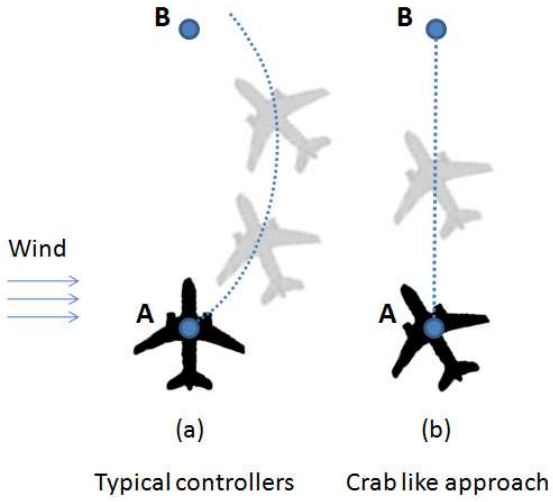


Figure 8.3: Heading hold vs Crabbing

and a disturbance observer/estimator is used to generate linear-adaptive guidance. Now PID controllers are linear by nature and have difficulties in the presence of non-linearities. On the other hand, the forces and moments produced by a vehicle's aerodynamic control surfaces are often nonlinear functions of control surface deflection [62]. The effect of crosswinds worsens this non-linear nature of the control response by adding the component of uncertainty. One of the methods used to tackle the problem of non-linearity is to use gain-scheduling, wherein the gains associated with the linear PID controller are scheduled (i.e. changed) based on a parameter. The parameter determines the operating region and as a result determines the gains to be used. In the context of winds, the parameter that determines the operating region is the wind speed and direction. However, we know that measuring wind speed and direction accurately on an aircraft is very hard using existing technology. We propose a reactive approach to the control problem without the need for wind speed/direction measurements. The following sections in this chapter shall discuss our proposed solution to achieve precise waypoint navigation in the presence of crosswinds.

8.3 Proposed controller overview

The function of the controller is to enable navigation of the UAV towards the waypoint commanded by the coordination component as shown earlier in Figure 7.3. We represent this target waypoint as point B, whose latitude and longitude are given by λ_B and ϕ_B. Let us also define λ_A and ϕ_A to be the latitude and longitude of A, the point at which the controller received the waypoint command. Finally, λ_C and ϕ_C shall be used to represent the latitude and longitude of point C, the current position of the UAV. We then obtain the cross-track distance, χ, as follows:

$$d_{A\to C} = \arccos\left(\sin\lambda_C \sin\lambda_A + \cos\lambda_C \cos\lambda_A \cos\left(\phi_C - \phi_A\right)\right) \tag{8.1}$$

$$d_{A\to B} = \arccos\left(\sin\lambda_B \sin\lambda_A + \cos\lambda_B \cos\lambda_A \cos\left(\phi_B - \phi_A\right)\right) \tag{8.2}$$

$$\psi_{A\to C} = \begin{cases} \arccos\left(\frac{\sin\lambda_C - \sin\lambda_A \cos\left(d_{A\to C}\right)}{\sin\left(d_{A\to C}\right)\cos\lambda_A}\right) & \text{if } \sin\left(\phi_C - \phi_A\right) < 0 \\ 2\pi - \arccos\left(\frac{\sin\lambda_C - \sin\lambda_A \cos\left(d_{A\to C}\right)}{\sin\left(d_{A\to C}\right)\cos\lambda_A}\right) & \text{if } \sin\left(\phi_C - \phi_A\right) \geq 0 \end{cases} \tag{8.3}$$

$$\psi_{A\to B} = \begin{cases} \arccos\left(\frac{\sin\lambda_B - \sin\lambda_A \cos\left(d_{A\to B}\right)}{\sin\left(d_{A\to B}\right)\cos\lambda_A}\right) & \text{if } \sin\left(\phi_B - \phi_A\right) < 0 \\ 2\pi - \arccos\left(\frac{\sin\lambda_B - \sin\lambda_A \cos\left(d_{A\to B}\right)}{\sin\left(d_{A\to B}\right)\cos\lambda_A}\right) & \text{if } \sin\left(\phi_B - \phi_A\right) \geq 0 \end{cases} \tag{8.4}$$

$$\chi = \arcsin\left(\sin\left(d_{A\to C}\right)\sin\left(\psi_{A\to C} - \psi_{A\to B}\right)\right) \tag{8.5}$$

The meaning of χ is illustrated in Figure 8.4. We propose the use of this above computed cross-track distance, χ, as the parameter for lateral control as opposed to the heading. The aim of the controller is now to asymptotically minimize χ to 0. The block diagram of the controller we propose for this purpose is shown in Figure 8.5.

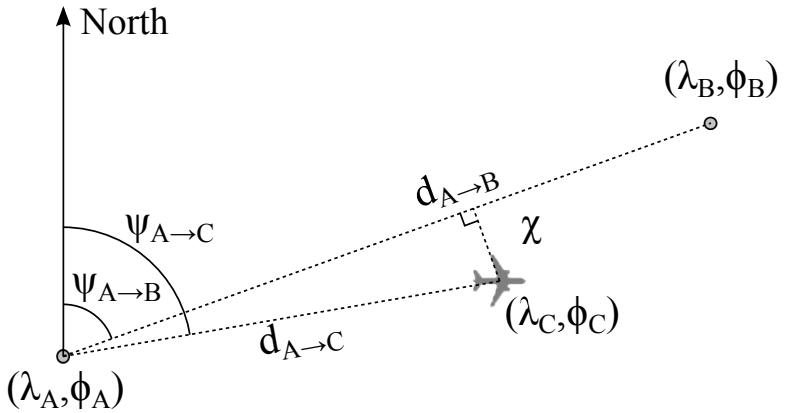


Figure 8.4: Cross-track distance, χ

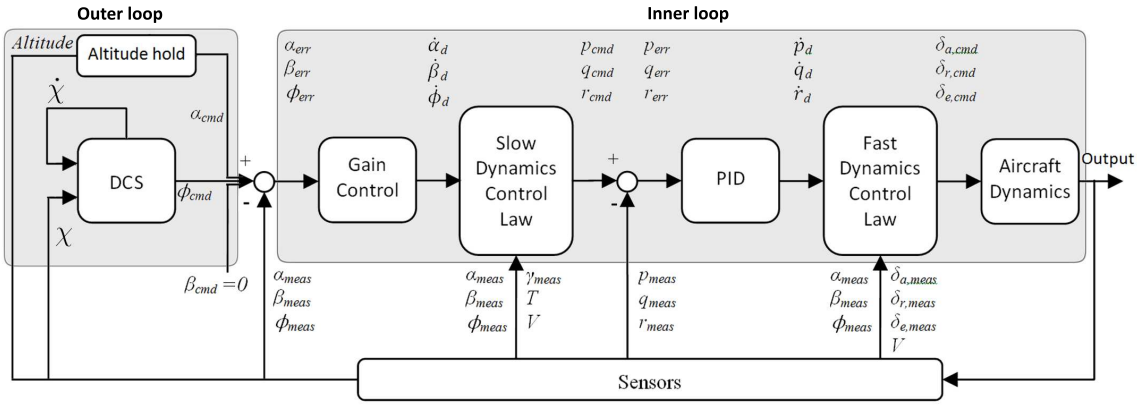


Figure 8.5: Dynamic Cell Structure (DCS) based Lateral Controller

8.4 Inner loop control

We treat the aircraft as a second order nonlinear system. The inner loop is a classical roll/pitch/yaw-tracking non-linear dynamic inversion (NDI) controller as shown in Figure 8.5. It uses the plant information along with the roll, pitch and yaw commands received from the outer loop to generate control surface deflections. Dynamic inversion is a technique used to provide desired dynamic response to a control system. It has been widely used in flight control systems because of the usefulness of its highly effective feedback linearization system. Works that have used the NDI controller for the inner loop include NASA's Intelligent Flight Control System (IFCS) [63], space-to-air re-entry vehicle flight control [64], etc. Isidori provides a detailed explanation and background knowledge on dynamic inversion in [65]. Essentially, it allows us to abstract away lower level, detailed dynamics. The equations of motion employed in the NDI implementation are as follows:

$$\begin{bmatrix} \delta_e \\ \delta_a \\ \delta_r \end{bmatrix}_{cmd} = \begin{bmatrix} 0 & L_{\delta_a} & L_{\delta_r} \\ M_{\delta_e} & 0 & 0 \\ 0 & N_{\delta_a} & N_{\delta_r} \end{bmatrix}^{-1} \left\{ \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}_d - \begin{bmatrix} 0 & L_\beta & L_p & 0 & L_r \\ M_\alpha & 0 & 0 & M_q & 0 \\ 0 & N_\beta & N_p & 0 & N_r \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ p \\ q \\ r \end{bmatrix}_{meas} - \begin{bmatrix} I_x & 0 & -I_{xz} \\ 0 & I_y & 0 \\ -I_{xz} & 0 & I_z \end{bmatrix}^{-1} \begin{bmatrix} I_{xz} p_{meas} q_{meas} + (I_y - I_z) q_{meas} r_{meas} \\ I_{xz}({r_{meas}}^2 - {p_{meas}}^2) + (I_z - I_x) p_{meas} r_{meas} \\ -I_{xz} q_{meas} r_{meas} + (I_x - I_y) p_{meas} q_{meas} \end{bmatrix} \right\} \tag{8.6}$$

where I_{ij} = moment of inertia about axes x, y, and z

L = aerodynamic rolling moment

M = aerodynamic pitching moment

N = aerodynamic yawing moment

p = roll rate

q = pitch rate

r = yaw rate

α = angle of attack

β = side slip angle

δ_e = aileron deflection angles

δ_a = elevator deflection angles

δ_r = rudder deflection angles

Equation 8.6 reflects the two modifications we have made to the classic NDI controller in order to make roll-tracking more accurate:

1. We identify the singularity problem associated with an ineffective control matrix just as in [64]. In order to bypass this problem, a two-time scale approach has been employed. The multiple time-scale approach separates the NDI controller into a fast dynamics control part and a slow dynamics control part. The fast dynamics portion handles variables such as pitch rate, roll rate and yaw rate, which respond quickly to control surface deflections. The slow dynamics portion on the other hand, handles variables such as angle-of-attack, sideslip and bank angle, which respond slowly to control surface deflections.

2. We also replace the classic gain controller that is normally used to obtain $\dot{p}_d$, $\dot{q}_d$, $\dot{r}_d$, from p_{err}, q_{err} and r_{err} respectively, with a PID controller. This

modification shortened the convergence time of χ to 0, as compared to a design using classic gain controller.

Although the NDI is capable of providing very accurate roll/pitch/yaw-tracking, matrix inversion can be rather computationally intensive. In our experiments, we test performance using both, the traditional PID, and the NDI for the inner control loop. The results are presented in Section 8.6.

8.5 Outer loop control

Our novel contribution towards the control design mainly lies in the outer loop. For the outer loop shown in Figure 8.5, we use a class of neural networks known as Dynamic Cell Structure (DCS) for lateral control. The altitude hold component, which is a PID controller, also forms part of the outer loop control. The purpose of the DCS is to obtain the appropriate roll angle command, Φ_{cmd}, based on the current χ regardless of wind conditions. We shall now take a look at the DCS, the modifications made to it and the training process used.

8.5.1 Dynamic Cell Structure (DCS)

The original DCS was proposed by Bruske and Sommer in [66]. It is essentially an RBF neural network with lateral connections between neurons. In their work, Bruske and Sommer chose an RBF representation to concurrently learn and use perfectly topology preserving feature maps (PTPM). The network applied Hebbian learning to adjust topological connections and a Kohonen-like learning rule to adjust node positions during training [66]. A graphical representation of the DCS is as shown in 8.6. Formally, the DCS network is defined as follows [67]:

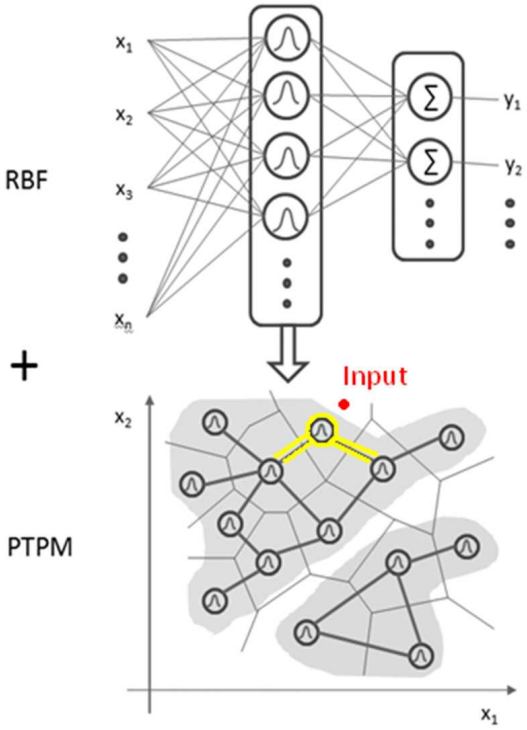


Figure 8.6: Dynamic Cell Structure [66]

Given an input manifold $I \subset \Re^n$ and output manifold $O \subset \Re^m$, with n usually $> m$, a DCS network is a collection of points:

$$n = (c, w, R, y) \in I \times O \times [0, 1] \times \Re_{\geq 0} \tag{8.7}$$

where : c = center of a point in $\Re^n$

w = weight in I associated with the point

R = function weighting the influence of n as a function of distance

y = error value associated with the estimation value of n as a graph

$$G = (N, L, S) \tag{8.8}$$

where : $N \subset I \times O \times [0,1] \times \Re_{\geq 0}$ is a set of nodes

$L \subset \{\{a,b\} \mid a,b \in N, a \neq b\}$ are links between nodes

$S : L \rightarrow \Re_{\geq 0}$ is a lateral connection strength function of an adjacency matrix C

$C \in \Re^{|N| \times |N|}$ for which:

$C_{ii} = 0 \; \forall \; i \in \{1,2,3,\ldots,|N|\}$

$C_{ij} = 0 \; \forall \; i,j$ not connected

$0 \leq C_{ij} \leq 1$ if i,j connected with strength C_{ij}, and

A connection strength function S is defined using a Hebbian rule:

$$C_{ij}(t+1) = \begin{cases} max\{y_i, y_j, C_{ij}(t)\} & : \; y_i y_j \geq y_k y_l \; \forall \; (1 \leq k,l \leq N) \\ 0 & : \; C_{ij}(t) < \theta \; \forall \; (1 \leq k,l \leq N) \\ \gamma C_{ij} & : \text{ otherwise} \end{cases} \tag{8.9}$$

where :

$\gamma \in (0,1)$ is a forgetting constant

$\theta \in (0,1)$ is a deletion threshold for weak lateral connections

$y_i = \dfrac{\|\boldsymbol{u} - w\|}{\sum_{i=1}^{n} \|\boldsymbol{u} - w_i\|}$ is the normalized distance to $\boldsymbol{u}$

where, w_i is the center of a neuron's receptive field and

$\boldsymbol{u}$ is a training pattern's coordinate location in $I \subset \Re^n$

For a given input, the best matching unit (bmu) is the center that is closest in distance to this input. Learning uses a standard Kohonen-like rule in which w_{bmu} and its topological neighbors are adjusted according to:

$$\Delta w_{bmu} = \varepsilon_{bmu}(\boldsymbol{u} - w_{bmu}) \tag{8.10}$$

$$\Delta w_{Nh(j)} = \varepsilon_{Nh(j)}(\boldsymbol{u} - w_{Nh(j)}) \tag{8.11}$$

where $Nh(j)$ of unit j is defined as

$$Nh(j) = \{i|(C_{ji} \neq 0, 1 \leq I \leq N)\} \tag{8.12}$$

Only the best matching unit and only its connected topological neighbors are adjusted during each learning cycle. Bruske and Sommer used insertion of nodes based on accumulated error [68]. They followed Martinetz's use of competitive Hebbian learning to adjust connection strengths while at the same time preserving topological mapping constraints. Nodes are added incrementally to the network during training after every ρ iterations (ρ is chosen arbitrarily). They are selected from areas with maximum estimation error. New nodes are placed between a node having the highest error and its topological neighbor having the second highest error. The location of the center of the receptive field for a new node w_{new} is calculated according to a ratio of error values $e_{w_{max}}$ and $e_{w_{2^{nd}_{max}}}$ where w_{max} and $w_{2^{nd}max}$ are the nodes with the highest and second highest error in the topological neighborhood respectively. The error of units w_{max} and $w_{2^{nd}max}$ are redistributed among w_{new}, w_{max} and $w_{2^{nd}max}$. The equations defining this distribution are as follows:

$$\begin{aligned}
\text{Define} \quad : \quad e_1 &= e_{w_{max}} \\
\text{Define} \quad : \quad e_2 &= e_{w_{2^{nd}_{max}}} \text{ then,} \\
r &= \tfrac{e_1}{e_1+e_2} \\
\Delta e_1 &= \tfrac{(1-r)e_1}{2} \\
\Delta e_2 &= \tfrac{e_2 r}{2}
\end{aligned}$$

And finally we get,

$$w_{new} = w_{max} + r(w_{2^{nd}max} - w_{max}) \tag{8.13}$$

$$e_{w_{new}} = \Delta e_1 + \Delta e_2 \tag{8.14}$$

$$e_{w_{max}} = e_{w_{max}} - \Delta e_1 \tag{8.15}$$

$$e_{w_{2^{nd}max}} = e_{w_{2^{nd}max}} - \Delta e_2 \tag{8.16}$$

These equations redistribute error equally among the three nodes and reduce overall error level in the region of maximum deviation from the tree function distribution. The connection strengths between the new node and the other two nodes are then adjusted. This is accomplished by setting:

$$\begin{aligned} C_{new,2^{nd}max} &= 1 \\ C_{new,max} &= 1 \\ C_{max,2^{nd}max} &= 0 \end{aligned}$$

The description thus far, has talked about the network of nodes and interconnections. Additionally, there is also an output, o_i, attached to every node to facilitate supervised learning. For a given input, $\boldsymbol{u}$, the output is given by,

$$Output(\boldsymbol{u}) = \sum_{i \in \{bmu \cup Nh(bmu)\}} a_i o_i \tag{8.17}$$

where, a_i is the activation of neuron i, on stimulus $\boldsymbol{u}$

$$a_i = \frac{1}{\sigma \|\boldsymbol{u} - w_i\|^2 + 1} \tag{8.18}$$

with $\sigma > 0$, representing the size of the receptive fields.

For every input received, the output associated with the *bmu* and its neighbors

are modified based on a delta rule. The delta rule below is derived by attempting to minimize the error in output through gradient descent.

$$\Delta o_j = \eta a_j \times (\text{Error in output}) \tag{8.19}$$

where, η is a constant that represents rate of descent.

8.5.2 DCS Training

The DCS is used for lateral control primarily due to its ability to map the behavior of multiple systems while being able to differentiate between them (through lateral neuron connections) and interpolate between them (neural network characteristic). The idea is to use a few systems, each catered to specific constant wind speed situations and have the DCS learn all of their behaviors. The systems that generate the data for the DCS will have to handle the situation assigned to each of them, successfully. For this purpose, the DCS is replaced by a PID in the controller shown in 8.5. So the PID takes χ as its input and produces a Φ_{cmd} as output to achieve the target of $\chi \rightarrow 0$. Four sets of gains are generated for each of the wind speeds of 0, 10, 20, and 30 kts. This is possible because wind speed can be set to a constant on the simulator and the gains can be tuned for that particular scenario. Each gain set is used to generate data for the DCS by running it in its corresponding fixed wind scenario. Each data point thus generated, consists of 3 values:

1. χ = cross-track error
2. $\Delta\chi = \chi(t) - \chi(t-1)$
3. Φ_{cmd} = commanded roll angle

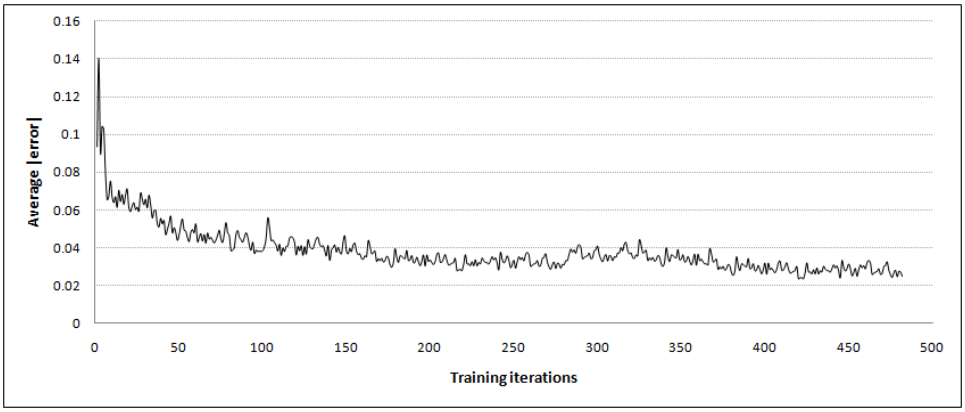


Figure 8.7: Average $|error|$ against training iterations (original DCS)

The first two, χ and $\Delta\chi$, serve as inputs to the DCS, while Φ_{cmd} is the output produced by the DCS. When a 3 valued data point is presented to the DCS, it learns what output it should be producing given that particular input. The number of input parameters is kept minimal and restricted to the important ones in order to reduce the dimensionality of the network. Considering that the controller would eventually be implemented on an embedded system with resource constraints, it is important to maintain low dimensionality.

The final data set consisting of 32748 inputs is then used to train the DCS through supervised learning. For training purposes, the DCS constants are set as follows:

Kohonen constants,	ε_{bmu}	$=$	0.2
	ε_{Nh}	$=$	0.2
Connection decay,	γ	$=$	0.99993
Connection threshold,	θ	$=$	0.01
Rate of gradient descent in output,	η	$=$	0.3

After every iteration of training, the network is tested against the same data set. The magnitude of error is averaged and plotted as shown in Figure 8.7. The network learned by the original DCS at the end of 480 iterations consisted of 3029 neurons.

8.5.3 DCS modifications

The output of the DCS is Φ_{cmd}, the scaled target roll angle, and 0.01 equates to 1 degree in roll angle. So an error of 0.03 was unacceptable in terms of accuracy. To increase accuracy and reduce training time, 3 changes were made to the DCS:

1. New rule for neuron addition: We noticed that the original DCS added a neuron every ρ iterations between the two neurons with the highest accumulated error. This prevented the network from reaching the outer edges of the input space. The network expanded very slowly using the Kohonen rule. To overcome this problem, a new neuron is added when the activity of the *bmu* (given by the radial basis function of the distance between the weights of the node and the input) is not sufficiently high and the error in estimation is not sufficiently low. In particular, when the network receives an input, whose distance from the *bmu* is higher than a threshold of 0.1, and whose estimated output has an error higher than a threshold of 0.1, a new neuron is added at the received input point. New connections are made between the new neuron and its *bmu* and the $2^{nd}bmu$. The output associated with this new neuron is set at one-third the desired output owing to a neighborhood of 3. As a result, the network grows faster and reduces the average error much faster.

2. Incorporate neighborhood information in delta rule: The original delta rule shown in Equation 8.19 does not take into consideration, the number of neighbors involved in changing the output. In other words, whether there were 2 or 20 neighbors, a given node would increase its output by the same number. The new proposed delta rule is given as follows:

$$\Delta o_j = \frac{a_j \times (\text{Error in output})}{(\text{Number of neighbors of } bmu)} \tag{8.20}$$

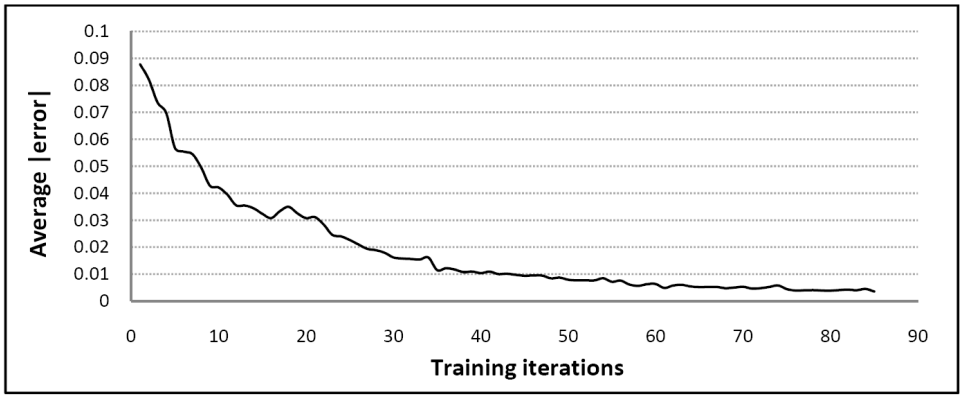


Figure 8.8: Average $|error|$ against training iterations (modified DCS)

3. Hebbian decay constant: The hebbian decay constant is determined using

$$\gamma = \theta^{\frac{1}{2n}} \tag{8.21}$$

where θ is the deletion threshold and n is size of dataset.

With the above modifications, the DCS is trained again with the same initial dataset. The magnitude of error is averaged and plotted as shown in Figure 8.8. As can be observed, the average $|error|$ now falls to approximately 0.003 in just 50 iterations. The number of neurons in the network at the end of 85 iterations is 2942, which is a little less than the number using the original DCS. Effectively, the modified DCS learned a 10 times more accurate representation in about $\frac{1}{6}$th the time.

8.6 Experimental results

8.6.1 Setup

The X-plane 8.64 simulator is used for testing controller performance. X-Plane is chosen primarily because of its modeling accuracy achieved by a process known as

"blade element theory" which breaks the aircraft down into many small elements and then finds the forces on each little element multiple times each second. The pre-loaded aircraft chosen for experiments is the PT-60 RC Plane which very closely models our real UAVs: Hangar 9 Alpha 60 and a scaled down version of the Pilatus PC-6 Porter. Multiple experiments are run with different wind profiles for each controller. In particular 5 controllers are compared against each other:

1. DCS-NDI
2. DCS-PID
3. PID-NDI with separately tuned outer loop PID for each wind speed scenario
4. Cascaded PID with separately tuned outer loop PID for each wind speed scenario
5. X-Plane Autopilot (Standard baseline)

At the beginning of each experiment, the aircraft is placed at a fixed waypoint **A** with a heading towards waypoint **B**, which is 2km away from **A**. Waypoint **B** is set as the target for the aircraft. A crosswind is then introduced perpendicular to the line segment AB (refer Figure 8.4). After about 65s, by which time χ is settled at 0, wind direction is switched 180 ° from original direction. The wind direction switch while keeping wind speed constant essentially tests the step response of the controller. The cross-track error, χ, i.e. perpendicular distance of the aircraft from line segment AB, is recorded through the course of each experiment.

8.6.2 Results and discussion

Figure 8.9 shows the performance of the various controllers at wind speeds of 10kts, 30kts, and 50kts.

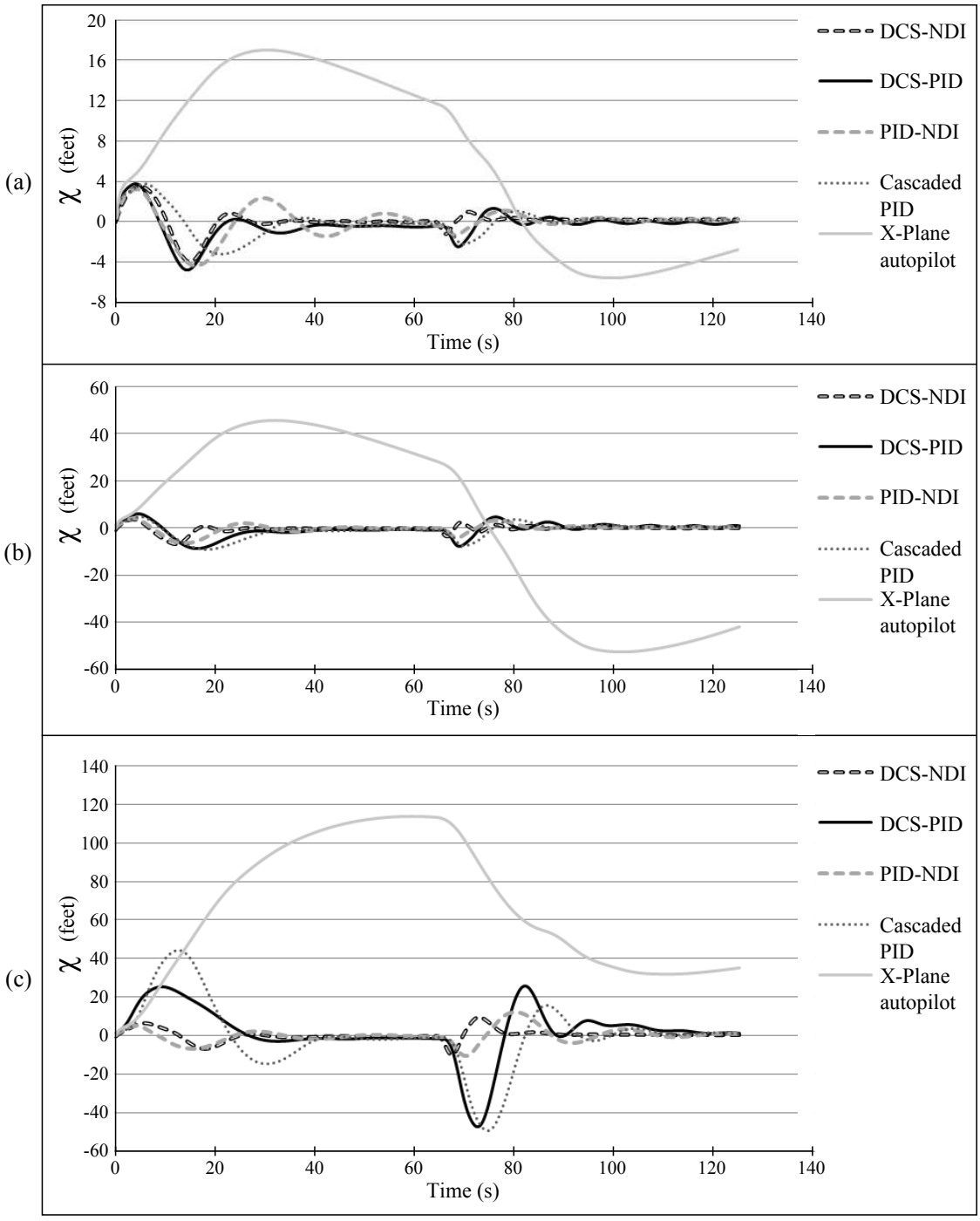


Figure 8.9: Controller performance comparison at wind speeds of (a)10kts, (b)30kts, and (c)50kts

Tables 8.1 and 8.2 show the maximum and average values of cross-track error achieved by the different controllers.

Table 8.1: Comparison of maximum cross-track error for various controllers

Wind Speed	Maximum χ				
	DCS-NDI	DCS-PID	PID-NDI	Cascaded PID	X-Plane Autopilot
50 kts	**10.0**	47.5	12.1	49.5	113.5
30 kts	6.8	8.9	**6.3**	9.3	52.5
10 kts	4.0	4.7	4.3	**3.8**	17.0

Table 8.2: Comparison of average cross-track error for various controllers

Wind Speed	Average χ				
	DCS-NDI	DCS-PID	PID-NDI	Cascaded PID	X-Plane Autopilot
50 kts	**1.8**	8.3	2.3	10.6	66.8
30 kts	**0.9**	2.1	1.1	2.2	35.4
10 kts	**0.6**	0.8	1.0	0.9	9.2

We observe that typical linear approaches tend to have higher maximum cross-track error, χ, and higher settling times. In particular, performance of the baseline PID controller (X-Plane autopilot) deteriorates substantially with increase in wind speed. The two controllers that used the DCS for the outer loop outperformed the other controllers. The average cross-track errors in the cases of both, the DCS-NDI and DCS-PID controllers, are around 90% lesser than that of the X-Plane autopilot for all wind speeds tested. The DCS-NDI controller has the lowest average cross-track error at all wind speeds, which can be attributed to its faster convergence to the desired flight path (i.e. line segment AB). It is closely followed by the PID-NDI, the DCS-PID and the dual-PID in that order.

The DCS effectively accounts for the non-linear nature of flight control response under crosswind conditions. The difference between using a traditional PID and an NDI for the inner loop is found to be minimal. The NDI does provide an increase in performance owing to dynamic gain scheduling. In fact, for our final control and coordination experiments, the NDI is used for the inner loop. However, in light

of the computational requirement, a PID for the inner loop might be the feasible choice when implemented for real flight.

The experimental results on simulation have thus proven the feasibility and the effectiveness of a reactive control system to account for wind effects without the need for wind speed and direction measurements. In that respect, our approach has the upper hand in comparison to "wind-frame" based methods [54, 55]. They also eliminate the need for restrictive mechanisms to handle wind at higher layers [56]. Our approach makes it possible for the higher layer coordination mechanism to assume precise waypoint navigation. A discussion of the controllers' performance on real flight tests is presented in Chapter 10.

8.7 Summary and Contributions

We presented a control system that uses the idea of crabbing to correct for crosswind effects. A dual loop controller was designed with the outer loop consisting of a dynamic cell structure (DCS) to generate roll angle commands for the inner loop. The inner loop consisted of a nonlinear dynamic inversion (NDI) component to generate appropriate control surface deflections. The DCS in particular was modified to enhance learning accuracy and speed without an increase in number of neurons. Experiments showed that the proposed controller (and a few variations) managed to substantially outperform a standard autopilot.

The main contribution of this chapter is a reactive control system capable of achieving accurate waypoint navigation despite adverse crosswind effects. The control component presented in this chapter introduces a novel system that works reactively using the normally-unused cross-track parameter along with a neural network, as opposed to existing solutions that require hard-to-obtain measurements

of wind speed and direction. It allows for realistic implementation of other higher level coordination algorithms that use waypoint navigation but do not consider wind effects.

Chapter 9

Multiagent Coordination

Chapter 8 covered the control component of the solution. In this chapter we look at the coordination component that makes higher level decisions and commands the control component. We tackle the combined problem of coverage, search and tethering as described earlier in Chapter 6. In particular, the decision-making process at each UAV and method of interaction and information exchange between UAVs so as to coordinate their effort towards search and relay, is presented in this chapter.

9.1 Related work

The combined problem of coverage, search and tethering as such is a new one. The sub-problem that has received the least attention is tethering whereas coverage and search have been studied to a good extent. We shall now look at related work in these areas.

9.1.1 Coverage and Search

The problems of coverage and search are very closely related. The coverage problem requires the deployment of stationary or mobile agents with the ultimate aim of maximizing the sensing area of the robots combined [69]. The problem of search requires the exploration of an area by mobile agents seeking to find one or more targets. The search problem usually includes the need for completing the search in a minimum amount of time or with a minimal overlap of covered areas by different agents. The search problem when combined with the coverage problem requires an exploration of the entire area. Further, in the case of persistent search, the search operation can never come to an end.

The earliest works on coverage dealt with the problem of planning the motion of a single agent in order to completely explore a given area. This was when the idea of grid based discretization of an environment was introduced. The approach of dividing an area into a grid of cells has become the most commonly used method for representing a search area [70]. In [71], Ge and Fua focus on limiting repeated coverage of areas that have already been explored by multi-robot teams. They do this by maintaining some unexplored regions around all of the explored areas and obstacles. They provide an upper bound for the time required to complete the task of exploring the area. However, in the worst case, this upper bound is the same as that of the single-robot scenario.

In [70], Kong et al. present an algorithm for dynamic exploration of an environment with obstacles using multiple agents. Information about new discoveries by one agent, including obstacles and cell-accessibility information, is communicated to other agents via a commonly shared adjacency graph. However, the main drawback of the solution provided is the assumption of unlimited communication capabilities. Rekleitis et al. [72] extend the single robot coverage algorithm to multiple robots.

Although they restrict communications to line-of-sight connections, they assume no limitations on communication range.

Many papers addressing coverage and search have looked at it as a sensor placement problem. They assume not all areas of the environment can be covered, and try to achieve information-theoretic goals such as minimizing entropy of possibly some parameter that needs to be estimated in the environment. One of the key contributors in this area is Guestrin et al. [73, 74], who exploit the concept of submodularity and locality in determining a near-optimal sensor placement plan.

Other theoretical works on search have dealt with robot path planning, where the paths are planned *a priori* with minimal adaptability. For example, in [75], Simmons et al. present a greedy heuristic algorithm to plan paths of robots capable of keeping the robots well separated. They do not consider communication capabilities of agents and thus have no information exchange. The resulting solution is a rigid one without the ability to handle much uncertainty.

Decentralized Multiagent Partially Observable Markov Decision Processes (Dec-POMDP) are a popular model of multiagent systems with uncertainty [76]. Dec-POMDP-based techniques have also been discussed in the cooperative search community. These approaches use probabilistic models of states and state transitions and observations. Usually rewards are assigned to actions and the task is to find a policy of actions that maximizes the expected reward over a finite time horizon [77]. The downside to these approaches is that a probabilistic model of the environment is required, which is not always available. Also, the computational complexity of finding a policy is high, especially with multiple agents and many state variables. Even in works as recent as 2008 [77], computation of the plan is centralized, while only execution is decentralized.

9.1.2 Tethering

In the tethering domain, much emphasis has been given to techniques for maintaining a fully connected network of agents at all times. For example, in [78], a chain of UAVs is maintained at all times so that a given UAV may communicate with any other UAV using multi-hop ad hoc routing. Basu et al. [79] take a similar approach and propose a flocking mechanism to maintain a swarm of UAVs interconnected and move them to the centroid of ground nodes. Correspondingly, numerous ad hoc routing protocols have been proposed for rapidly changing network configurations [80]. However, often there are not enough UAVs to establish a continuous link between two points on the ground and this is a problem for solutions that require a fully connected UAV mesh.

The notion of a continuous link between end-points is meaningful when the relays are stationary. Mobile relays on the other hand, can act as ferries to deliver packets, thus eliminating the need for a continuous link. In order to support such mobile relays, a new class of routing protocols have emerged for what are known as Delay Tolerant Networks (DTNs). The concept of delay tolerance is directly applicable when using UAVs as communication relays. Research on DTNs has been extensive over the last few years. However, little work has been done on cooperatively controlling UAVs to physically establish such DTNs.

A few fixed trajectory solutions have been proposed that utilize DTNs for establishing communication between mutually unreachable ground nodes. All methods discussed in Section 3.2, namely SRT, NRA [23], MRT [30], DARD [31], FRA [23], l-SCFR [32], and FRA [23], fall under this category. The one work that addresses a similar problem and specifically uses UAVs is that by Frew et al. [81]. However in their work, Frew et al. tackle the case with only two ground nodes. They use one of two configurations: chain-relay or conveyor belt [81]. However, neither is

directly scalable for scenarios with a number of ground nodes scattered in a given area. In fact, all these fixed trajectory solutions would cease to work with the introduction of mobile ground nodes. Moreover, if communication is to be established for survivors too, searching the area would be a necessity that cannot be achieved with fixed trajectory solutions.

In one of the most recent works in this area, Liu et al. [82] study the use of POMDPs to generate policies for the single agent scenario. They assume a stochastic model for node mobility and try to generate a policy for the single agent that ferries packets between these nodes. The policy is used for finding a node within a given small portion of the entire area. The policy for choosing which node to visit however is predetermined.

9.2 Assumptions

The expanded problem definition was earlier stated in Chapter 6. Given the challenges discussed in Section 6.1, the only assumptions we make in trying to solve the problem are as follows:

1. Localization: Every UAV is assumed to be equipped with a GPS receiver for position information. The error in GPS position estimate is considered to be negligible with respect to inter-node distances and is not explicity addressed.

2. Communication: Each UAV is also mounted with an omni-directional WiFi antenna with a transmitting power of 20dBm, which gives a theoretical communication range of $\sim 350m$. We assume a practical communication range of 150m air-to-ground and 200m air-to-air.

3. Lower layer capability: The coordination component assumes the presence of an accurate waypoint navigation layer below. This has been covered in Chapter 8.

9.3 Coordination Architecture

The coordination architecture we propose is a distributed one, wherein each agent makes decisions independently, using information from own sensors and from communicating with neighboring agents. Each agent's belief of the world is represented using two data structures:

1. An integer matrix, $\mathbb{T}_k$ for every agent k: We call it the Visit Map as it holds timing information for last visit for all grid cells. Essentially,

$$\forall(i,j), \mathbb{T}_{k_{ij}} = \text{ time elapsed since any agent last visited grid cell } (i,j)$$

This is different from the belief map that many other papers on multi-agent target search use. In many works, the belief map is a probability distribution giving the probability of finding a target in a given grid cell[83]. We however use elapsed time in order to enable the hybrid state where an agent performs the search operation as well as relaying of packets between ground nodes. The behavior of an agent in this hybrid state is detailed in section 9.4.3. The use of $\mathbb{T}_k$ also makes data fusion easier as it has age information embedded in it. The update rule for the matrix with time is as follows:

$$\forall(i,j), \mathbb{T}_{k_{ij}}(t+1) = \begin{cases} \mathbb{T}_{k_{ij}}(t) + 1 & \text{if } (i,j) \neq x_k(t) \\ 0 & otherwise \end{cases}$$

where $x_k(t)$ is the the current position of agent k

As a result, the value for each matrix cell is incremented by 1 at every time step. When the agent flies over a particular grid cell (i, j), the corresponding value for that matrix cell is reset to 0.

2. A set of coordinates, $\mathbb{G}_k$, held by agent k, containing currently known positions (in grid coordinates) for ground nodes, called the Position List: This set, $\mathbb{G}_k$, is updated when the agent k detects a wireless signal from a ground source when flying over a grid cell. It is also updated when information is received from neighboring agents.

The behavior of an agent using this belief information is determined by an adaptive finite state machine, described in Section 9.4. Belief information is exchanged between agents when they come within communication range of each other. Both, $\mathbb{G}_k$ and $\mathbb{T}_k$ are exchanged between neighboring agents. Received data is fused with local belief information at the Environment Estimator block in Figure 7.3. The details of what entails an information exchange is provided in section 9.5.

9.4 Adaptive Finite State Machine

Every agent, at its core is a behavior-based control system. At the higher level though, every agent is capable of operating in one of the following 4 states: search (SR), relay (RL), search and relay hybrid (HB), and proxy (PR). The adaptive finite state machine (FSM) as shown in Figure 7.3 is responsible for deciding the operational state of an agent. The behavior of an agent within a given state would be a part of the Cooperative Path Planner block in Figure 7.3. We now take a look at the behavior of an agent in each of the 4 states and how the adaptive FSM makes its decisions.

9.4.1 Search State

In the search state, an agent flies around the entire grid looking for wireless signals from a ground source. We do not use predetermined search patterns like those mentioned in [84] so that UAVs can be added or removed from the multi-agent system at any time. This is essential because in a realistic setting, UAVs may run out of power or fuel, or may suffer failures in mid-air that require their withdrawal and replacement. In other words, an agent's decision cannot depend on the knowledge of number of UAVs on the field. Moreover, since ground nodes can emit signals intermittently, the search operation can never come to an end. In other words, the agents need to revisit each grid cell repeatedly through time. Keeping these requirements in mind, we aim to design a cooperative search mechanism where behaviors of each agent combine to provide an emergent behavior wherein the multiagent system spreads out to search the entire area.

Traditional decision-theoretic approaches try to find a near-optimal policy using a stochastic model of the environment. They try to maximize a reward over a finite horizon in order to come up with the decision policy. However it is not always possible to obtain an *a priori* stochastic model, as is the case in our problem. There is absolutely no knowledge of where and for how long ground nodes might appear. We take an approach that is "near-decision-theoretic", in the sense that agents try to maximize a scoring function with the aim of increasing cell visit frequencies. The scoring function is chosen such that it can also be used for the purpose of lowering packet latency. At every instance when an agent moves from one grid cell to another, it recomputes its action. When an agent wants to compute its action, it applies a scoring function to every cell in the grid as shown in Equation 9.1. The destination, y_k, is then determined by picking the cell with the highest score.

$$\zeta_k(i,j) = w_{k_t}\mathbb{T}_{k_{ij}} + w_{k_h}H(h_{k_{\text{pref}}} - h_{k\to ij}) + w_{k_d}F(d_{k_{ij}} - d_{k_{opt}}) \qquad (9.1)$$

$$y_k = \arg\max_{(i,j)} \ \zeta_k(i,j) \tag{9.2}$$

In the above equation, $\zeta_k(i,j)$ is the score assigned by agent k to grid cell (i,j). It is computed as a summation of three components. The first is $w_{k_t}\mathbb{T}_{k_{ij}}$ where w_{k_t} is a positive weight. This term represents the desire of each agent to visit the grid cell that has not been visited in the longest while, i.e. the grid cell with the highest value in the matrix, $\mathbb{T}_k$. This is essential because the validity of information about a cell decays with time. As a result, the cell with the highest elapsed time, is the one about which there is least certain information. The second term in Equation 9.1 is $w_{k_h}H(h_{k_{\text{pref}}} - h_{k\to ij})$ where w_{k_h} is a positive weight and $h_{k\to ij}$ is the heading from x_k to the center of grid cell (i,j). $h_{k_{\text{pref}}}$ refers to the preferred heading of agent k and is given by

$$h_{k_{\text{pref}}} = \begin{cases} h_{k_{curr}} & \text{if } N(k) = \emptyset \\ \frac{1}{|N(k)|}\sum_{i\in N(k)} h_{i\to k} & \text{otherwise} \end{cases} \tag{9.3}$$

$$\text{where} \quad h_{i\to k} \text{ is heading from agent } i \text{ to agent } k$$
$$N(k) = \{i| \text{ agent } i \text{ is in range of agent } k\}$$

Equation 9.3 means that the preferred heading of an agent is its current heading, unless there are neighboring agents within communication range. Essentially, every agent desires to move forward where forward is defined as any direction falling within $\frac{\pi}{4}$ radians from the current heading as can be visualized in Figure 9.1. In the presence of neighbors, the preferred heading of an agent is the average of the set of headings away from every neighboring agent. This mechanism mainly achieves the spread of agents in opposite directions. On the other hand, it also achieves collision avoidance if the communication range is higher than the minimum turn radius. However, if packet losses occur, collision avoidance cannot be guaranteed. One viable option to ensure collision avoidance is to use redundant long distance radios along with algorithms that have been proposed to guarantee collision avoidance

using multiple altitudes [85]. The difference between $h_{k_{\text{pref}}}$ and $h_{k \to ij}$ is fed as input to function, $H(x)$, which is a combination of two Gaussian functions given by

$$H(x) = \begin{cases} e^{-\frac{x^2}{2}} & \text{if } |x| \leq \frac{\pi}{4} \\ 5e^{-\frac{x^2}{c}} & \text{if } |x| > \frac{\pi}{4}. \end{cases} \tag{9.4}$$

$$\text{with} \qquad c = \frac{2\pi^2}{32 \ln 5 + \pi^2}$$

As a result, the second term in Equation 9.1 has the highest value when the difference between $h_{k_{\text{pref}}}$ and $h_{k \to ij}$ is 0 and gradually decreases until the magnitude of this difference hits $\frac{\pi}{4}$. When the magnitude of the difference increases beyond $\frac{\pi}{4}$, the value of the second term in Equation 9.1 drops steeply towards 0. All in all, the second term represents the desire of an agent to continue flying along the preferred heading (preferably within $\frac{\pi}{4}$ radians of $h_{k_{\text{pref}}}$) which would be plain forward in the absence of neighboring agents and away from all neighboring agents, if there were any.

The third term in Equation 9.1 represents the fact that each agent in general concerns itself with the cells closest to it and gives them more importance as compared to cells that are farther. It is given by $w_{k_d} F(d_{k_{ij}} - d_{k_{opt}})$ where w_{k_d} is a positive weight and $d_{k_{ij}}$ is the distance between x_k and the center of grid cell (i, j). $d_{k_{opt}}$ is given by

$$d_{k_{opt}} = \sqrt{2} r_k \tag{9.5}$$

$$\text{where} \qquad r_k = \text{ minimum turn radius of agent } k$$

In order to understand the reason behind Equation 9.5, let us imagine a situation where agent k is at the center of grid cell $(5, 5)$ and heading towards $(5, 6)$. The score for grid cell $(5, 7)$, $\zeta_k(5, 7)$, would be high even though it might be impossible to reach $(5, 7)$ with a simple bank maneuver owing to the minimum turn radius

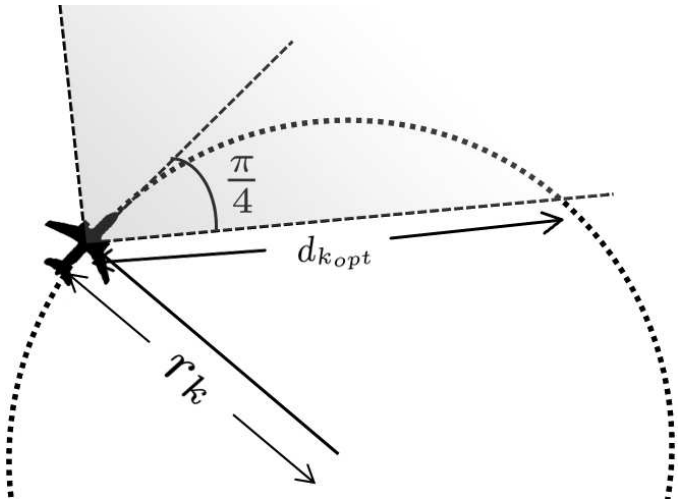


Figure 9.1: Optimal distance, $d_{k_{opt}}$

of a UAV. As a result the optimal distance to look ahead for an agent should be the distance of the intersection of the UAV's trajectory with maximum bank angle and the line emanating from the agent at an angle of $\frac{\pi}{4}$ from the current heading as shown in Figure 9.1. The angle of $\frac{\pi}{4}$ is chosen so as to fall in line with the definition of forward. The difference between $d_{k_{ij}}$ and $d_{k_{opt}}$ is fed as input to the function $F(x)$, which is a Gaussian function given by

$$F(x) = e^{-\frac{(x)^2}{2\sigma^2}} \tag{9.6}$$

$$\text{with} \qquad \sigma = \frac{d_{max}}{\sqrt{2\ln(100)}}$$

where d_{max} is the maximum possible distance between two points on the grid (the diagonal if it is a rectangle). As a result, the third term peaks when $d_{k_{ij}} = d_{k_{opt}}$ and drops as the difference between the two increases.

The three weights w_{k_t}, w_{k_h}, and w_{k_d} are chosen based on the following rules:

1. Considering cells (i_1, j_1) and (i_2, j_2) at a fixed distance from the agent, if $h_{k_{\text{pref}}} - h_{k \to i_1 j_1} = 0$ and if $h_{k_{\text{pref}}} - h_{k \to i_2 j_2} = \frac{\pi}{4}$, then a difference in elapsed time of $\mathbb{T}_{k_{i_2 j_2}} - \mathbb{T}_{k_{i_1 j_1}} = C_t$ should make the scores for both equal.

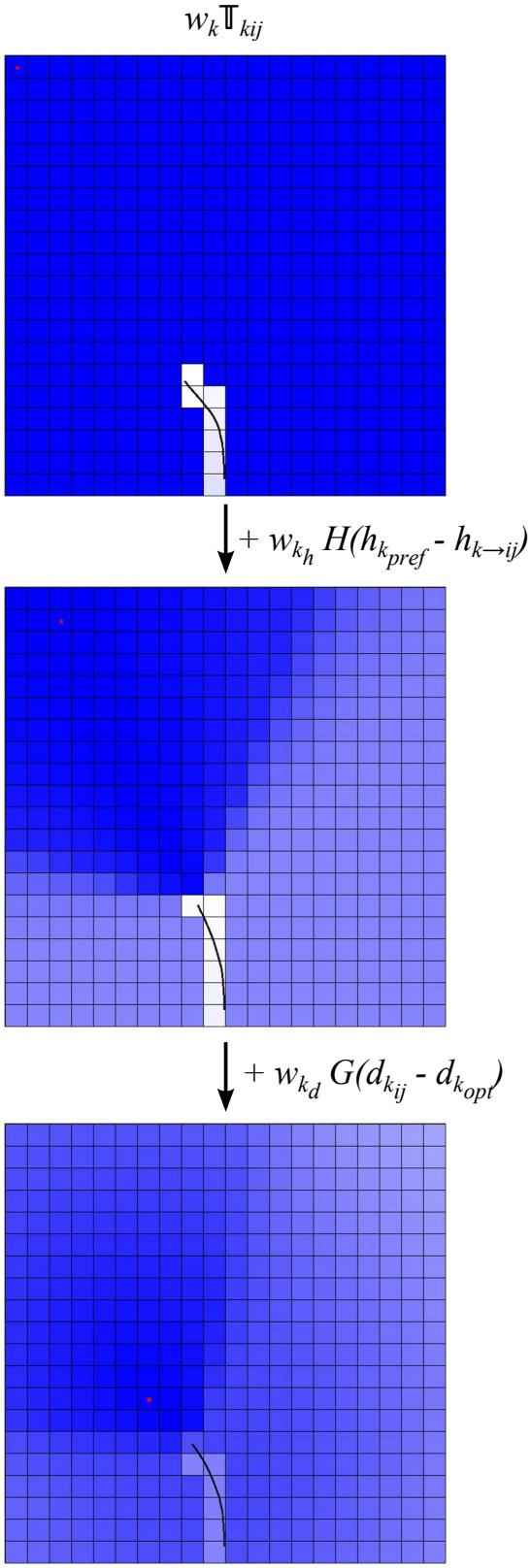


Figure 9.2: Scoring function applied incrementally - darker cell represents a higher score and the cell with the red dot represents the chosen cell

2. Considering cells (i_1, j_1) and (i_2, j_2) with the same elapsed time value in $\mathbb{T}_k$, if $h_{k_{\text{pref}}} - h_{k \to i_1 j_1} = 0$ and $h_{k_{\text{pref}}} - h_{k \to i_2 j_2} = \frac{\pi}{4}$ and $d_{k_{i_2 j_2}} = d_{k_{opt}}$, then a difference in distance of $d_{k_{i_1 j_1}} - d_{k_{i_2 j_2}} = C_d$ should make the scores for both equal.

Using these 2 rules and setting $\forall k(w_{k_t} = 1)$, we get

$$w_{k_h} = \frac{C_t}{1 - e^{-\frac{\pi^2}{32}}} \qquad \text{and,} w_{k_d} = \frac{C_t}{1 - e^{-\frac{C_d^2}{2\sigma^2}}}$$

C_t and C_d are constants that have an explicit meaning as defined in the rules above and can be given values based on preference. For the experiments discussed in Section 9.6, we use C_t =100s and C_d =500m.

The effects of each of the components in the scoring function (Equation 9.1) when applied incrementally is shown in Figure 9.2. The figure considers the grid as seen right at the start when the agent sets out. The cells are shaded to represent their relative scores. Darker the cell, higher the score.

9.4.2 Relay State

In the relay state, the agent does not concern itself with the search operation and dedicates itself to relaying packets between ground nodes. Ideally, agents in the RL state should follow paths generated by the solution to BECDLMST (Chapters 4 and 5). However, BECDLMST is a centralized solution requiring knowledge of all ground node locations and number of agents involved. In a decentralized scenario with limited knowledge, each agent is likely to have a different view of the environment (i.e. set of ground nodes and their locations, states of other agents). Generating the same tree at each agent would be improbable. As a result we choose to employ a method that tries to partially mimic the BECDLMST.

In devising a decision method for RL agents, we take a look at the chain-relay architecture, which is a fixed trajectory latency minimizing solution for 2 ground nodes, discussed by Frew et al. in [81]. The chain relay architecture is illustrated in Figure 9.3.

Figure 9.3: Chain-relay architecture

Firstly, we notice that BECDLMST can be viewed as an extension of the chain-relay architecture applied to more than 2 ground nodes. In the case of 2 ground nodes, the decision method that would give rise to the chain-relay architecture is as follows:

1. Move towards ground node A until ground node A or another agent is encountered
2. Move towards ground node B until ground node B or another agent is encountered
3. Go back to step 1

We extend this idea to the case of multiple ground nodes in order to approximate an extended chain-relay architecture, i.e. the BECDLMST. As opposed to simply switching between two ground nodes, the multiple ground node scenario will require agents to put in more thought when deciding their next target ground node. Here we apply the same idea as in the SR state. When it comes to deciding on the next action, the agent applies the scoring function, $\zeta_k(i,j)$, only to ground node cells, i.e. $(i,j) \in \mathbb{G}_k$. As a result, the agent picks the ground node that obtains the highest score and heads towards it. If an agent k were to meet any other agent m in the RL or HB state (i.e. $m \in N_{RL,HB}(k)$), it immediately resets $\mathbb{T}_{k_{ij}} = 0$, for

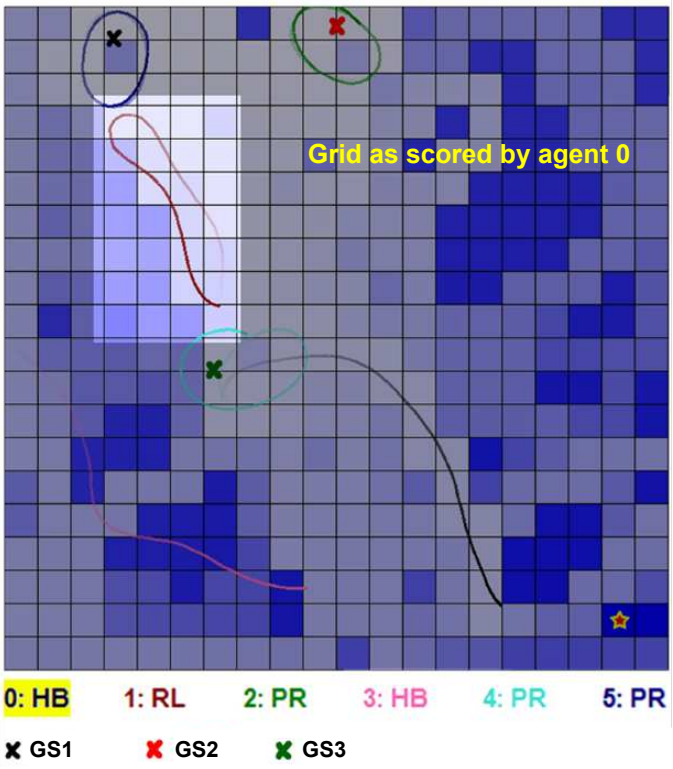


Figure 9.4: Example of agent in relay (RL) state

the ground node cells that are closer to the neighboring agent and recomputes its action. As a result, the following proposition holds true.

$$\forall(i,j) \in \mathbb{G}_k \left(\exists m \left(m \in N_{RL,HB}(k) \wedge d_{m_{ij}} < d_{k_{ij}}\right) \rightarrow \mathbb{T}_{k_{ij}} = 0\right) \tag{9.7}$$

The above rule is applied because the agent that is closer to a given ground node should be in charge of delivering packets to that ground node, and there is no point in sending more than one agent, moving together, towards the same ground node. As an effect of this rule, the emergent behavior is a multi-node extension of the latency minimizing chain-relay architecture illustrated in Figure 9.3. An example of an agent in the RL state is shown in Figure 9.4.

9.4.3 Hybrid Search and Relay State

In the hybrid search and relay state, the agent performs the search operation as well as relaying of packets between ground nodes. Every time the agent needs to compute its next action, it scores all cells in the entire grid using the formula in Equation 9.1. In other words, it performs all the steps laid out in section 9.4.1. However, when computing scores for ground node cells, the first term corresponding to elapsed time is given a lot more importance. In particular, $w_{k_t}\mathbb{T}_{k_{ij}}$ becomes $w_{k_t}(\mathbb{T}_{k_{ij}})^2$ so as to represent the higher visit frequency requirement of ground node cells. When a ground node cell gets chosen based on highest score, the agent implicitly works on relaying packets. In a similar manner to the RL state, whenever the agent gets within communication range of another agent in RL or HB state, the rule in Equation 9.7 is applied.

9.4.4 Proxy State

In the proxy state, the agent moves in a circular motion with minimum turn radius over the ground node for which it acts as proxy. Having a proxy for every ground node is essential in order to maintain the DTN protocol implementation on the agents, transparent to the ground node. Equipment held by survivors and rescuers would in most likelihood use standard IP networking protocols. IP is not designed to be delay tolerant and lacks the key features of a DTN protocol: buffering, and opportunistic forwarding [6]. IP would simply drop all packets if no route existed to the destination. The proxy agent is used to stop IP from doing that by letting the ground node know that a route exists. On receiving packets from the ground node, the PR agent would only need to buffer all received packets and forward them when an RL or HB agent comes by. As a result of using proxy agents, the service time, s_g in Equation 6.1, is maximized for each ground node, g, that has a

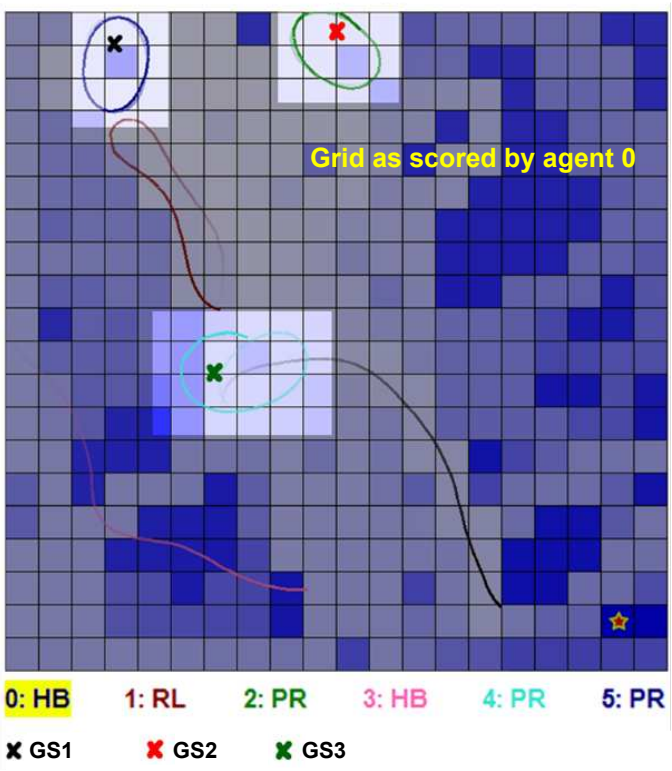


Figure 9.5: Example of agents in proxy (PR) state

proxy. The other task of the agent in PR state is to keep track of the ground node it is in charge of. While flying in circles, the agent tries to maintain connectivity with the ground node at all times. It maintains an estimate of the ground node's position and circles around this point. Therefore, if it ever discovers that a part of the circle is beyond the range of the ground node, it updates the position estimate by moving it a small distance directly away from the arc that fell out of range. The proxy agent for a given ground node has complete authority over the position information for that ground node. It is the only agent allowed to make changes to the position estimate for that particular ground node, and when the estimate changes, the proxy agent informs the change to all agents that pass by. Figure 9.5 shows examples of agents in the PR state.

Every proxy agent also maintains the average visit frequency for the ground node over the past T_W time units. If this average visit frequency drops below a threshold, $\nu + \Delta\nu$, the agent decides to recruit an RL agent. Once the decision to recruit is

made, the first agent that comes into contact with the PR agent is recruited as an RL agent. If recruitment is unsuccessful even after $T_{recruit}$ time units, the PR agent decides to personally forward the packets to the next closest known proxy agent (or base station) and informs the other agent to recruit an RL agent. The PR agent then returns to its corresponding ground node and waits. If at any point the average visit frequency goes above $\nu + \Delta\nu$, the PR agent decides to dismiss an RL agent. The minimum time period between two consecutive recruitments or dismissals is T_W.

Since ground nodes can be intermittent, proxy agents continue circling for a fixed period of time, T_C, when the ground node disappears. If the agent does not receive any wireless signal from the ground node within T_C, the agent considers the possibility that the ground node might have moved. It then begins spiraling outwards up to a distance beyond which the ground node could not have traveled given a maximum speed of 10ms^{-1}. If the ground node is still not found, the agent assumes the ground node is lost and switches its operational state to HB. It then spreads the information about the missing ground node to other agents, thus causing them to delete the corresponding element in $\mathbb{G}_k$.

9.4.5 State transitions

The state transitions are based on the state diagram shown in Figure 9.6. All agents start out in the HB state, which is equivalent to the SR state when no ground nodes have been found. An agent k in the HB state decides to switch to the SR state, if $\max_{(i,j)} \mathbb{T}_{k_{ij}}$ becomes higher than $\rho + \Delta\rho$. A high value in the $\mathbb{T}_k$ matrix means the search is slow and the average visit frequency is low, thus requiring a more dedicated search effort. Once the dedicated search manages to increase the average visit frequency, the SR agent should be able to switch back to

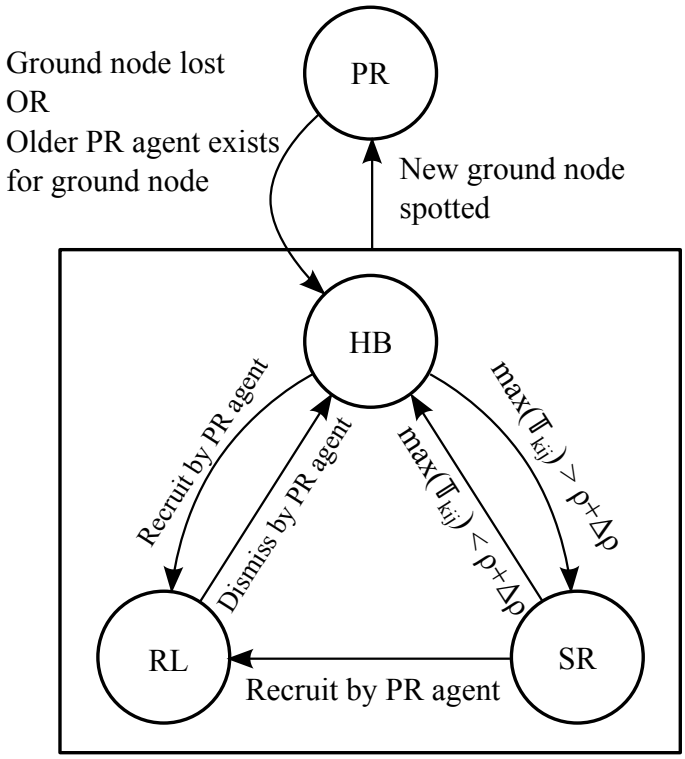


Figure 9.6: State Diagram

HB. The switch from SR to HB takes place when $\max_{(i,j)} \mathbb{T}_{k_{ij}}$ becomes lower than $\rho + \Delta\rho$.

The transition to the PR state can take place from any other state. When an agent spots a ground node that does not already have a proxy agent, the agent immediately switches to the PR state and becomes in charge of that ground node. If an agent mistakenly became the proxy agent of a ground node with an existing proxy agent, the newer one reverts to the HB state and gives way to the original proxy for that ground node. Otherwise, an agent in the PR state switches to the HB state only if the ground node has been deemed lost as per section 9.4.4. The only way any agent can enter the RL state, is being recruited by a proxy agent. The recruitment process has been discussed in section 9.4.4. The transition from the RL state to the HB state happens only when the RL agent gets dismissed by a PR agent.

Adaptive state transition

The thresholds for state transitions are represented as $\rho + \Delta\rho$ and $\nu + \Delta\nu$ so that ρ and ν can be kept constant while updating $\Delta\rho$ and $\Delta\nu$ to manipulate the thresholds. These thresholds indirectly determine the ratio of number of agents in the RL, HB and SR states. The ratios in turn affect the value of Q that we try to maximize. We reiterate the definition of Q here for easy reference:

$$Q = \frac{f_{avg} s_{avg}}{\tau} \tag{9.8}$$

We know that s_{avg} is maximized as a result of having proxy agents. Consequently, the two factors that remain variable are τ, and f_{avg}. τ can be reduced by increasing the number of RL agents. However, that would adversely affect the search operation and reduce f_{avg}. The key is to balance them out so as to maximize Q. However, latency is highly dependent on the relative positions of the ground nodes (possibly mobile), and average visit frequency is dependent on the number of agents on field, both of which are not known for certain and are dynamic. In other words, the values of $\rho + \Delta\rho$ and $\nu + \Delta\nu$ also need to be dynamic so as to update the ratios of agents in different states, appropriately.

The correct way would be to evaluate Q from time to time and adapt the threshold values. However, in order to obtain values for τ and f_{avg}, global knowledge would be necessary. To overcome this problem, we propose the use of estimates for Q derived at each agent. The estimates can then be passed on to the base station if and when the agent comes in contact with the base. Using multiple estimates, the base station can make an informed decision as to how to update the threshold values. The updated values can be disseminated through agents that pass by. To be able to produce an estimate for Q, the agents need to perform 2 additional tasks:

1. Maintain a set TS_{ij} for each grid cell that holds timestamps of all the times when the value in the cell drops. Any element in the set that has a timestamp earlier than T_W time units prior to current time is discarded. $\frac{|\mathrm{TS}_{ij}|}{T_W}$ then gives the visit frequency for cell (i,j) over the last T_W units.

2. Update the timing information on the DTN protocol header (this field is assumed on the header, but even otherwise it is a single integer that is added to the header) by adding the duration for which the agent held the packet.

As a result, every agent would have its own version of f_{avg}. The agent that delivers a packet to the destination would have latency information for that packet. Using the latency information for all packets delivered in the last T_W time units, the agent can generate an estimate for maximum latency, τ. If and when an agent k passes by the base station, it delivers $\hat{f}_{avg_k}$ and $\hat{\tau}_k$, which are estimates by agent k for f_{avg} and τ in Equation 9.8. An agent never modifies its trajectory with the aim of delivering these values to the base, because it is of lower importance than other operations and there is bound to be some agent that delivers packets to the base station that can provide its estimate. Finally, the two values can be used by the base to generate $\hat{Q}_k$, an estimate for Q. The base uses a reference value for the product of latency and average visit frequency. The reference value, φ, is calculated by the base station using its current knowledge of ground node positions. φ is given by

$$\varphi = \frac{\sum_{g1,g2 \in G, g1 \neq g2} d_{g1 \to g2}}{XY} \tag{9.9}$$

The idea is that τ should in general be proportional to the sum of distances between any pair of ground nodes, and inversely proportional to UAV speed and number of UAVs if all agents were involved in relaying packets. Similarly, f_{avg} should be inversely proportional to area of the grid and directly proportional to UAV speed

and number of UAVs if all agents were involved in the search operation. The product of the two should cancel out UAV speed and number of UAVs to give Equation 9.9. This is a combination of 2 individually optimal cases for latency and visit frequency. So the product is a reference value for the situation where equal numbers of agents are involved in each of search and relay operations. The base can then compute $\hat{f}_{avg}\hat{\tau}$ to get an idea of relative distribution of SR and RL agents. This information as well as the trend in Q can be used to obtain the update rule for $\Delta\rho$ and $\Delta\nu$ as follows:

$$b(\hat{Q}) = \text{ slope of regression line through estimates of } Q \text{ over past } T_W \text{ time units} \tag{9.10}$$

$$\Delta\rho(t) = \frac{\rho}{\varphi}\left(\hat{f}_{avg}\hat{\tau} - \varphi\right) + \Delta\rho(t-1)b(\hat{Q}) \tag{9.11}$$

$$\Delta\nu(t) = \frac{\nu}{\varphi}\left(\hat{f}_{avg}\hat{\tau} - \varphi\right) + \Delta\nu(t-1)b(\hat{Q}) \tag{9.12}$$

Updates for both ρ and ν are derived similarly. In Equation 9.11, the new $\Delta\rho$ depends on the previous $\Delta\rho$, as well as the deviation in the $f\tau$ product from the reference value (scaled to the same order as ρ), and finally, $b(\hat{Q})$, which determines whether to switch the sign of $\Delta\rho$.

9.5 Belief Information Exchange (Environment Estimator)

Belief information, i.e. $\mathbb{G}_k$ and $\mathbb{T}_k$ are exchanged when any two agents come within communication range of each other. It is important for every agent to know the locations of all ground nodes. This necessitates the full exchange of $\mathbb{G}_k$. When an agent receives a neighbor's list of ground node locations, it simply merges its own

list with the received list

$$\mathbb{G}_k = \bigcup_{i \in (k \cup N(k))} \mathbb{G}_i$$

As for $\mathbb{T}_k$, a full exchange would mean that each message would contain XY number of integer values, which would require a high bandwidth. In fact the bandwidth issue is a prominent one in the general cooperative target search problem. One approach to address it is suggested by De Lima et al. in [86] where they propose the exchange of only recently updated cell information. This very often turns out to be only those cells that lie in the recent flight path of the sender. Since any cell can be on the list, the sender would also need to include co-ordinate information for each cell whose information is sent, thus tripling the number of bytes required to represent the information for 1 cell. Some others have suggested the exchange of last known positions of other agents along with their destinations [87, 83]. However, this would require each agent to maintain the history for all other agents. Moreover, if the destination of an agent is chosen as a relatively close point to current location, as in our case, there is very little information embedded.

We propose the use of neural networks to estimate the values of grid cells given some intelligently chosen partial information. We utilize the general tendency of agents to move in straight lines and the minimum turn radius of UAVs to come up with a pattern of cells whose information would be sufficient to obtain a good estimate of the remaining cells. The pattern we propose is the one in Figure 9.7. The effect is that any straight path taken by agents would intersect with a shaded cell at least once every 5 cells. Moreover, the minimum turn radius of the UAVs wouldn't allow them to fly in a circle without cutting across a shaded cell. In reality though, no agent in a non-PR state would fly in circles.

Essentially, this pattern would require only $\frac{3}{8}$th the information for the entire grid. However, the packet size would increase linearly with respect to grid area and

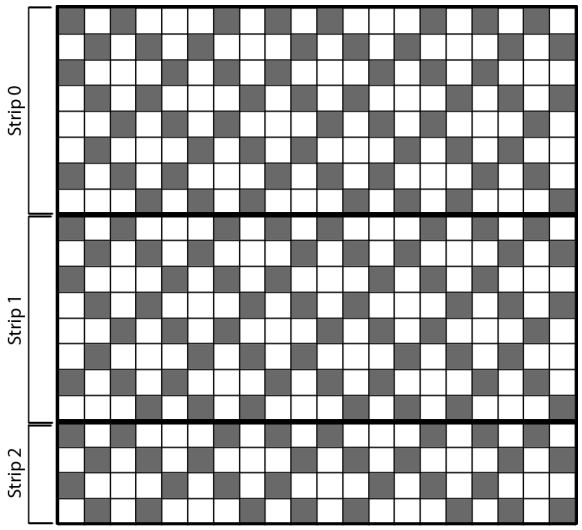


Figure 9.7: Pattern of cells chosen for exchange

that is quadratic with respect to a given side, if it is a square. To overcome this problem, we divide the grid into horizontal strips each 8 cells thick, as shown in Figure 9.7. Each strip has an index number starting from 0 for the first strip. When a neighboring agent sends a packet, it specifies the index number and a sequence of integers giving values corresponding to shaded cells alone. The co-ordinate information is not required because the receiving agent knows exactly which cell each value corresponds to in the fixed pattern. Since the thickness of each strip is fixed, packet size would increase linearly with respect to one of the sides of the grid. The sending agent would first send the strip that contains the receiving agent. This would have information pertaining to the immediate environment of the receiving agent. Following this, the sending agent sends strips that are increasingly far away from the receiving agent on either side (above or below). The last strip sent would be the furthest from the receiving agent and would be of least immediate relevance.

Having received a packet, the agent estimates the values for the remaining cells using Dynamic Cell Structures (DCSs), which are discussed in detail in Section 8.5.1. The DCS is chosen because it is capable of differentiating between different

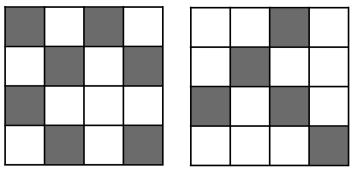


Figure 9.8: Types of blocks handled by each DCS

situations by using lateral connections between neurons that are related. The hypothesis is that the DCS should be able to distinguish between different cases when an agent flies across the block horizontally or vertically, etc., and interpolate correctly. In order to avoid the curse of dimensionality, we extract 4×4 blocks from the grid, each of which serves as a data point. We identify 2 types of 4×4 blocks where each pattern in a type is only a rotated version of the corresponding block shown in Figure 9.8(a) and 9.8(b). To be more specific,

$$\forall i, j \left((i+j) \text{ is even}\right), \quad \left\lfloor \frac{(i+j+2) \bmod 8}{4} \right\rfloor \equiv i (\bmod 2) \rightarrow B_{ij} \in \text{TYPE1}$$
$$\left\lfloor \frac{(i+j+2) \bmod 8}{4} \right\rfloor \not\equiv i (\bmod 2) \rightarrow B_{ij} \in \text{TYPE2}$$
$$\text{where,} \quad B_{ij} \text{ is the block with top left corner at } (i, j)$$

Therefore, DCS1, after learning, takes a 7-tuple as its input corresponding to the shaded cells in Figure 9.8(a) and produces a 9-tuple output corresponding to estimates of the unshaded cells. Similarly, DCS2 takes a 5-tuple as its input and produces a 11-tuple output. The data to train the two DCSs is obtained by running simulations with full communications (full exchange of $\mathbb{T}_k$). From one snapshot of the 20×20 grid on one agent, 73 data points are produced for DCS1 and 72 data points are produced for DCS2. The simulation is run for 30 minutes to generate millions of data points to train the DCS. Subsequent to supervised learning offline, DCS1 (93 neurons, 7% estimation error) and DCS2 (129 neurons, 7% estimation error) are used on every agent to estimate missing cells in information received from neighbors.

9.6 Simulation and Results

9.6.1 Centralized heuristic vs. decentralized RL state behavior

Our first set of experiments is designed to test the performance of agents in the RL state. The aim is to compare the performance of decentralized wireless backbone formation (using only locally exchanged information) to that of BECDLMST using the heuristic algorithm discussed in Chapter 5. We study the loss in performance when moving from an all-knowing oracle to a realistic scenario. JSBSim 1.0 is used to simulate the behavior of each aircraft. Each agent is a standalone instance of JSBSim running on a single node of a computing cluster. The cluster used for experiments consisted of 22 nodes, with a mixture of CentOS and Solaris based quadcore server-grade computers. Agents communicate with each other over the network. A separate compute node simulates ground nodes.

Setup

1. Area: 2km×2km divided into a 20×20 grid
2. Ground nodes: N nodes located randomly [stationary and non-intermittent]
3. Agents:
    - Maximum speed = 25ms^{-1}
    - Controller : DCS-PID
    - Every ground node cell has a PR agent [not counted as part of the M agents so that both cases have the same number of agents free to perform store-carry-forward routing]
    - Start from the center of bottom edge of grid

For the first set of experiments, we make the conditions in the simulation as similar as possible to the assumptions in the centralized heuristic algorithm. In particular the following three conditions are enforced:

1. Ground nodes are considered to be stationary and non-intermittent
2. All M agents are placed in the RL state from beginning to end [i.e. searching is not performed]
3. Communication range is assumed to be $\sim 10m$ so that it is negligible in comparison to the dimensions of the grid. We do not use the realistic communication range of 200m, because that would not be negligible in comparison to 2km and would result in a considerably lower latency for the decentralized simulation.

The number of ground nodes, N, and number of agents M are varied, and for each N, M combination, 20 runs of the experiment are conducted each with a random node configuration. The average for each N, M combination, using both simulation and the centralized heuristic, are presented in Table 9.1.

Table 9.1: Comparison of network latency using centralized heuristic and RL state simulation for various N, M combinations

	Network latency (seconds) for M=3			Network latency (seconds) for M=5		
N	Heuristic	RL	% difference	Heuristic	RL	% difference
3	134	162	20.9	124	148	19.4
5	292	417	42.8	191	278	45.5
10	605	837	38.3	292	449	53.8
20	1050	1328	26.5	498	704	41.4

	Network latency (seconds) for M=10			Network latency (seconds) for M=20		
N	Heuristic	RL	% difference	Heuristic	RL	% difference
3	95	111	16.8	81	90	11.1
5	142	187	31.7	127	139	9.4
10	224	325	45.1	166	201	21.1
20	265	389	46.8	249	296	18.9

The results show that decentralized coordination in the RL state leads to network latencies that are in the range of 10-50% higher than that achieved using the

centralized heuristic. We notice that in general, with an increase in the number of agents, the difference between results obtained the two methods tends to decrease. This can be attributed to the fact that in the decentralized method, agents tend to meet more often if there were more agents on the field. It should however be noted that the communication range is forced to be extremely low in the decentralized simulation so as to match the assumption in the centralized heuristic.

9.6.2 Experiments with complete system

Now we look at experiments to test performance of the complete system performing coverage, search and tethering. For all the following experiments, the entire system described in Sections 9.3-9.5 is used. Also, the communication range of the UAVs is now set to the realistic value of 150m air-to-ground and 200m air-to-air.

Effects of increasing agent count for same ground node setting

In this section, we perform experiments to study how the variation of agent count affects the network performance for a given ground node setting. Experiments for this purpose were conducted using a combination of 2 simulators: X-Plane 8.64 for realistic UAV control and Qualnet 4.5 for realistic communications. Any communication between 2 UAVs goes through Qualnet, which determines whether the wireless transmission was a success based on its communication model. UAVs in this simulation have a maximum speed of $25 ms^{-1}$.

We again use a 2km×2km disaster area for all experiments. The agents start out at random positions on the field with no prior knowledge of ground nodes except for the base station. Three ground nodes are used of which 1 is stationary and continuously emitting, while the other 2 are intermittent with one being mobile

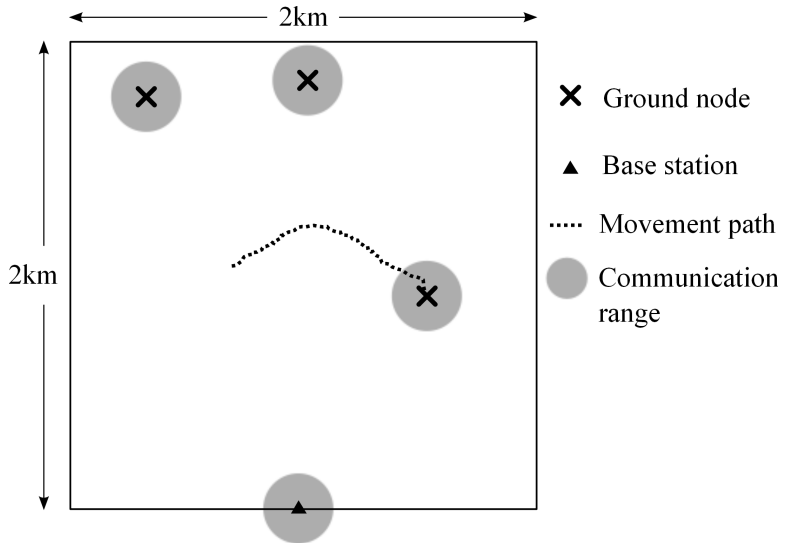


Figure 9.9: Positions of ground node and path of mobile ground node

and the other, stationary. The initial positions of all ground nodes and the path for the mobile ground node is as shown in Figure 9.9. The mobile ground node stops emitting wireless signals for very short durations. Ground node 2 on the other hand, only appears after 100s through the simulation and disappears again at 400s, before reappearing at 600s. Every ground node generates packet streams of 10 packets/second towards every other ground node, including the base station. Every experiment is run for 15 minutes and repeated 5 times to ensure no anomalous behavior. The parameter varied between each experiment is number of agents. We start with 5 agents so that there are at least 2 agents remaining when 3 of them become proxy agents. We increase the number of UAVs until 10. Figure 9.10(a) plots τ over time for one sample run of each experiment. The value for τ at any instance in time is the maximum latency of all packets delivered in the last T_W before the time instance. T_W is chosen to be time taken by an agent to traverse half the distance from edge to edge, which is 40s. We notice that τ spikes up from time to time, but the number of spikes and the height of the spikes reduces with an increase in number of agents. The spikes are caused by packets having to wait at source for longer durations when fewer agents are available. However, such long latencies would affect the value of ν thus causing more agents to be recruited into

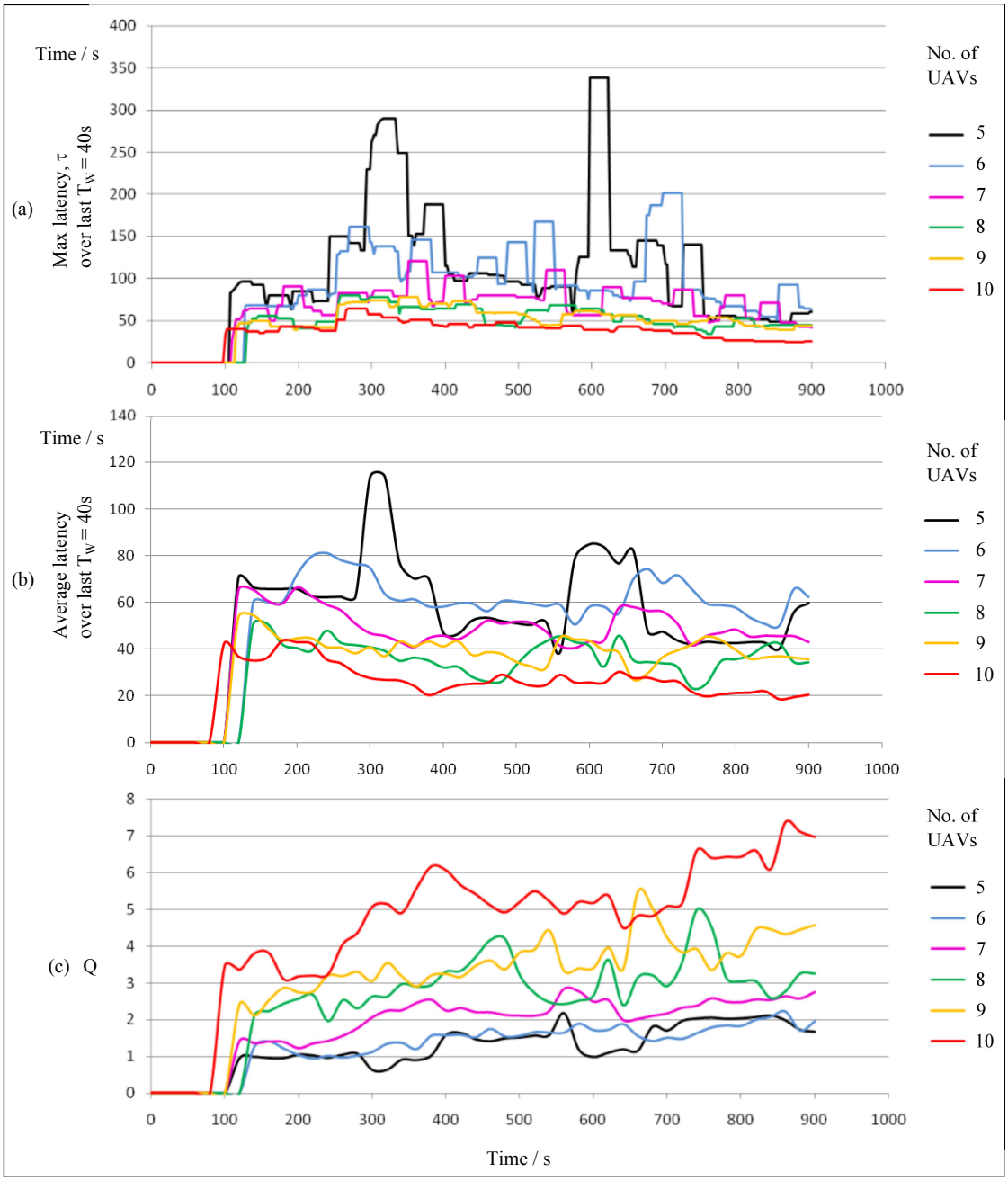


Figure 9.10: (a) τ, maximum latency of packets delivered in the last T_W time units (b) average latency of packets delivered in the last T_W time units (c) Q based on Equation 9.8

the RL state. As a result, maximum latency is lowered fairly quickly. In the case of lower values for M, we notice that the oscillation of latency values is higher. This can be attributed to the fact that agents tend to transform back and forth between the RL and SR state often in order to balance between the search and relay operations. With a higher number of agents, the balance is easier to achieve and more agents tend to stay in the HB state thus causing lesser oscillation in latency values.

We also plot the average latency over time to study how it differs from maximum latency through time. We observe the general tendency of latency to reduce with increase in number of agents. Also, average latency tends to be considerably lower than maximum latency. The interesting behavior though is the sudden spikes in latency at times when a new ground node is found. This can be attributed to the fact that the new PR agent would have begun buffering packets from the new ground node, but stayed unable to forward them until another agent came in contact with this PR agent. However, the adaptive state transition mechanism ensures that RL agents are introduced to minimize this latency. Within each experiment, we also observe that latency generally decreases. This is a direct consequence of trying to maximize Q in Equation 9.8, which is plotted in Figure 9.10(b) and shown to be generally increasing in value, albeit non-monotonically.

In Figure 9.11, we observe the relative distribution of agents in different states through time. We see how SR agents slowly become unnecessary with increase in number of agents because HB agents can already provide a satisfactory search effort. The consistently changing SR:HB:RL ratios we observe can be attributed to the effective adaptive state transition mechanism that modulates $\Delta\rho$ and $\Delta\nu$ to maintain a positive slope for Q.

The values of maximum $\mathbb{T}_{ij}$ from the global version of the visit map, are plotted in Figure 9.12. The longest any cell had to wait for a re-visit is very low for all

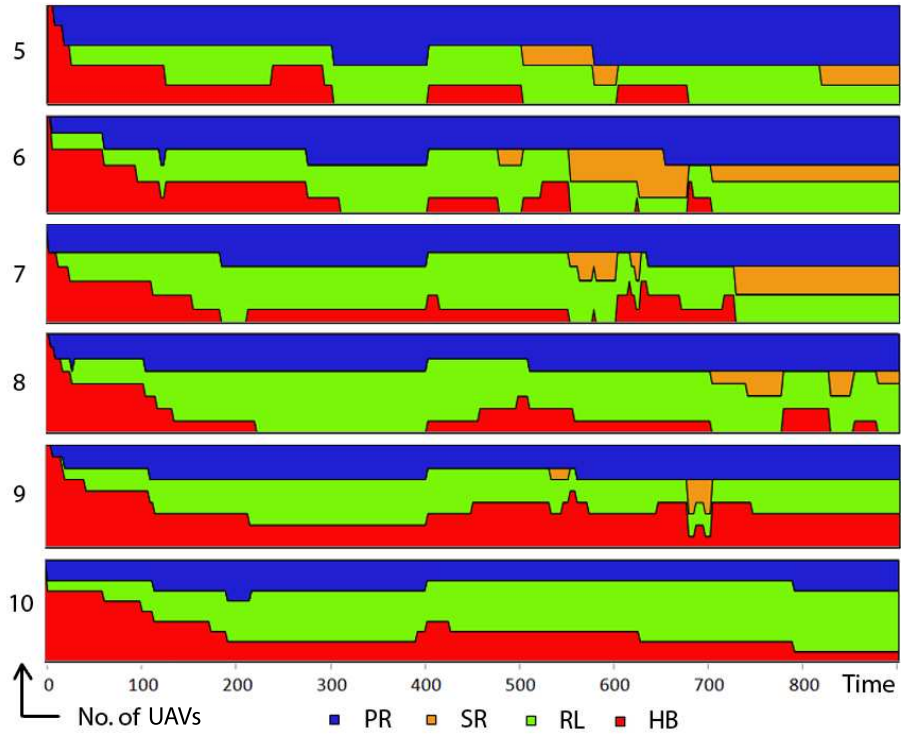


Figure 9.11: Distribution of UAVs in the 4 states through time for each experiment

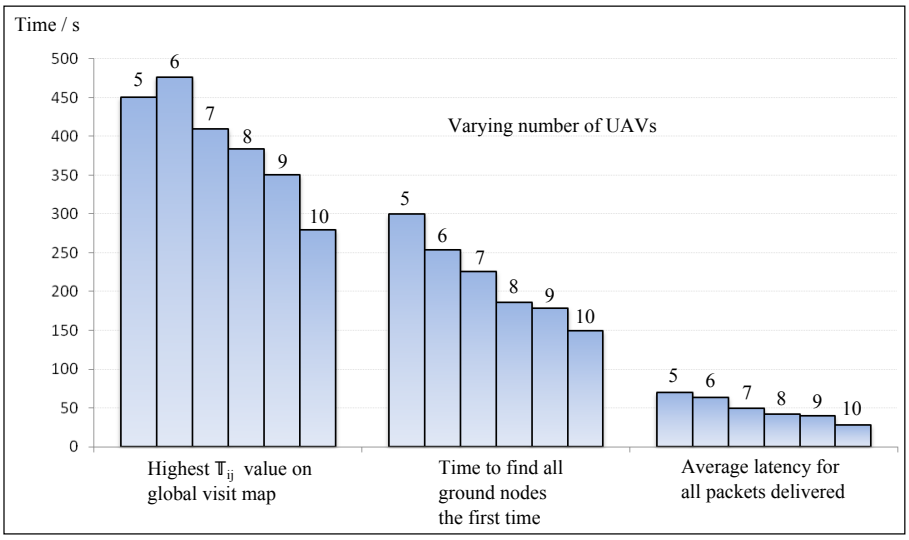


Figure 9.12: Global performance metrics (averaged over multiple runs of each experiment)

the experiments considering 3 of the agents would have been in PR state. The observation that actually goes to show the effectiveness of the search method laid out in section 9.4.1, is the time taken to find all ground nodes (shown in Figure 9.12). The spread of a search needs to be wide with minimal overlaps to have a low find time. The results here go to show that the scoring mechanism in Equation 9.1 achieves exactly this. Finally, a relatively low average network latency implies that the emergent network architecture is a latency minimizing one.

9.7 Summary and Contributions

We presented a novel cooperative control mechanism to coordinate a swarm of UAVs and establish a wireless communications backbone connecting multiple ground stations. In particular, 4 states of operation were introduced with an adaptive state transition mechanism. The adaptive update rule modified the behavior of the swarm based on the current state, so as to minimize packet latency and maximize cell visit frequency. The search operation was designed in such a way that the same data structures could be used for the relay operation as well, thus enabling a hybrid search and relay state. We also considered the bandwidth limitations of wireless links and presented a novel solution to acquire fairly accurate information from neighboring agents despite using little bandwidth (specifically $\frac{3}{8}$th). To this end, a DCS-based state estimation procedure was proposed that utilizes knowledge of UAV dynamics, restrictions, and the general behavior of an agent. Empirical results showed that the proposed mechanism was not only able to perform both the search and relay operation efficiently but also adapt to changing situations such as addition or loss of ground stations.

The main contribution of this chapter is an efficient solution for the novel problem of coverage, search and tethering, combined, under realistic wind conditions and

communication limitations. We provide a control and coordination solution that balances the tasks of search and relay, while minimizing latency and maximizing visit frequency. Importantly, this is achieved in a realistic setting with decentralized control, without global information at each agent, regardless of intermittency of ground stations, in the presence of winds, and under realistic communication limitations.

Another contribution of this chapter is the belief exchange mechanism proposed as part of the coordination algorithm that can be applied to many other applications using UAV swarms. The idea of using the limitations of UAV motion in choosing grid cells that encompass enough information to interpolate missing cell information is applicable to situations other than the specific problem considered in this thesis.

Chapter 10

Practical implementation and testing

In this chapter, we present flight tests conducted with real aircrafts to validate the control algorithms and the viability of air-air and air-ground communications.

10.1 Hardware

The hardware chosen for experiments were off-the-shelf equipment so as to assure ease of availability and reproduction. A number of concerns were addressed in choosing the appropriate airframe and the electronics that went inside, such as:

1. Payload capability of aircraft - need to be able to carry autopilot, separate onboard processor, wireless router and antenna.
2. Stall speed of aircraft to be kept low for safety
3. Weight of autopilot, sensors, onboard processor and other electronics to be kept low

4. Provide hardware-based failsafe for manual override

5. Autopilot code needs to be modifiable down to the control loop

10.1.1 Airframe

Two kinds of airframes were used in the tests conducted. One was chosen to be big enough for carrying all required payload (onboard processor, wireless router and antenna) for data communication. A different, much lighter airframe (i.e. more susceptible to winds) was chosen to test the performance of the control algorithms. Specifications of both airframes are provided below:

Pilatus PC-6 Porter Scale 150

A custom built scaled down version of the 1/150 scale Pilatus PC-6 Porter, called the Pilatus PC-6 Porter Scale 150 [88], was used for tests conducted to validate air-air and air-ground communication. This was the bigger of the two airframes mentioned above. Figure 10.1 shows the aircraft on the field. Below are the aircraft specifications:

- Length: $1.96m$
- Wingspan: $2.64m$
- Height: $0.57m$
- Wing area: $0.84m^2$
- Airfoil: NACA2415
- Wing loading: $9.76kg/m^2$
- Weight: $8.4kg$

Figure 10.1: Pilatus PC-6 Porter Scale 150

The airframe ensured sufficient space for loading all the required electronics. The payload capacity of the Pilatus is about 5kg which gives enough room for additional sensors and equipment. The propeller is powered by a gasoline engine with a 1 litre fuel tank that allows for half an hour of flight time.

Multiplex Mentor

The Multiplex Mentor is a lightweight foam aircraft with just sufficient space to hold the autopilot unit. This airframe was chosen to test the control algorithm, owing to its high susceptibility to winds. Figure 10.2 shows the aircraft on the field. Below are the aircraft specifications:

- Length: $1.17m$
- Wingspan: $1.63m$
- Wing area: $0.45m^2$
- Airfoil: Flat bottom high-wing placement

Figure 10.2: Multiplex Mentor

- Wing loading: $4.45kg/m^2$
- Weight: $2.0kg$

The propeller is powered by a 3-cell LiPo battery that allows for 10 minute flight time.

10.1.2 Autopilot unit

The Ardupilot Mega [89] microcontroller board and its accompanying ArduIMU Shield (sensor board) were used to implement our control algorithms. The reason for choosing the Ardupilot Mega is its functionality and availability of open-source support. The Ardupilot Mega controller board is shown in Figure 10.3. The Ardupilot Mega is based on the 16MHz Atmega1280 microcontroller. It has a processing power of up to 32MIPS. The onboard memory includes 128k of program flash, 8K SRAM and 4K EEPROM. A separate circuit with a multiplexer and an Atmega328 processor is used to transfer control from the RC system to the autopilot and back.

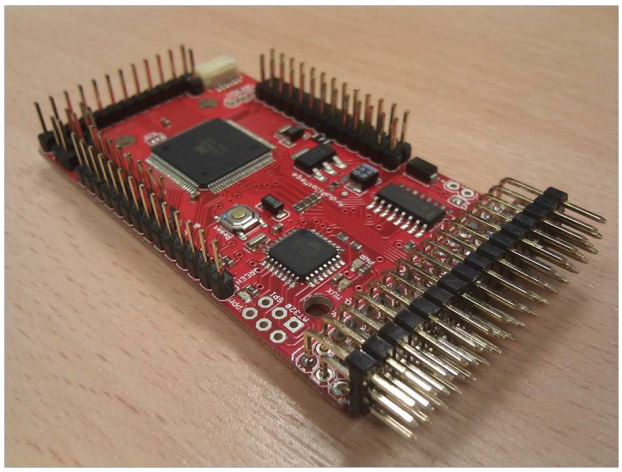


Figure 10.3: Arudpilot Mega controller w/ Atmega1280

Figure 10.4: ArduIMU Shield

The board's I/O capabilities include 16 analog inputs (sensors), 40 digital I/O pins, 4 serial ports, 8 RC channels, 8 PWM outputs, 1 6-pin (EM406 standard) GPS input.

The ArduIMU Shield holds a number of required sensors such as a triple-axis gyroscope, an ADX330 accelerometer and an Absolute Bosch pressure sensor (for altitude). A dedicated I2C channel allows the connection of several more sensors through a "Daisy chain board". The onboard 12-bit ADC is used for converting all analog sensor values to high precision digital values to be fed as input to the Atmega1280. The IMU shield also extends one of the serial ports to a USB port via an FTDI adapter. Figure 10.4 shows an ArduIMU Shield and figure 10.5 shows

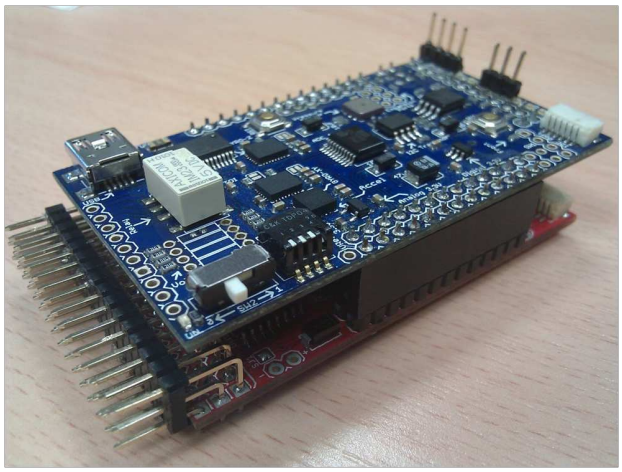


Figure 10.5: Ardupilot Mega controller connected to an ArduIMU Shield

the Ardupilot Mega controller connected to the IMU Shield.

10.1.3 External sensors

GPS unit

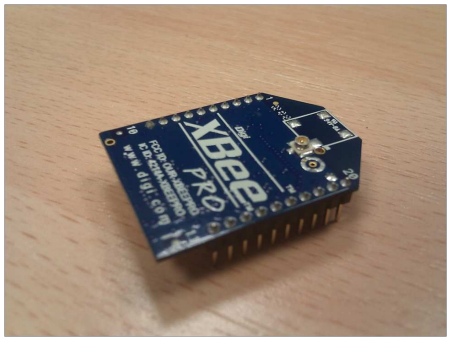
A GS407 Helical GPS Receiver is used for localization and navigation. The GS407 hosts a U-Blox LEA 5H chipset. The GPS unit provides a 2Hz positional update. It is connected to the Ardupilot controller through a U-Blox adapter that converts the GS407 connector to an EM406 connector. The GPS unit and the adapter are shown side by side in Figure 10.6.

Airspeed sensor

An MPXV7002 airspeed sensor along with the corresponding pitot-static tube setup is used to measure the forward speed of the aircraft with respect to air. The speed measurement is mixed with the speed measurement obtained from GPS data to derive the aircraft's speed. Figure 10.7a shows the airspeed sensor used onboard the aircraft.

Figure 10.6: GS407 Helical U-blox GPS Receiver and Adapter

(a) MPXV7002 airspeed sensor



(b) Xbee Pro 2.4GHz telemetry unit

Figure 10.7: Airspeed sensor and Telemetry unit

10.1.4 Telemetry

An XBee Pro 2.4GHz telemetry unit is used to relay real-time information to the base station that can be used to track the status of the aircraft. The radio has a 10mW output giving a theoretical line-of-sight range of 1 mile. These modules use the IEEE 802.15.4 networking protocol for communication. Figure 10.7b shows the telemetry unit.

Figure 10.8: Advantech PCM3386 embedded computer

10.1.5 Onboard computer and wireless equipment

We use an Advantech PCM3386 embedded computer for implementing the DTN protocol stack and interfacing between the wireless equipment and the autopilot. The PCM3386 is based on the PC/104 standard and has an Intel Celeron M 1GHz CPU. It is connected to the Ardupilot Mega via USB. A standard off-the-shelf 802.11g router is connected using ethernet. Figure 10.8 shows the PCM3386 board. We run a stripped down version of Linux on the PCM3386. The programs use serial communication to receive and send data to the autopilot. Standard socket programming is used to access the underlying implementation of the DTN protocol. The onboard computer and the network stack were implemented by our project collaborators. The implemented DTN protocol which uses Reliable UDP [90] is detailed in their publication at [91].

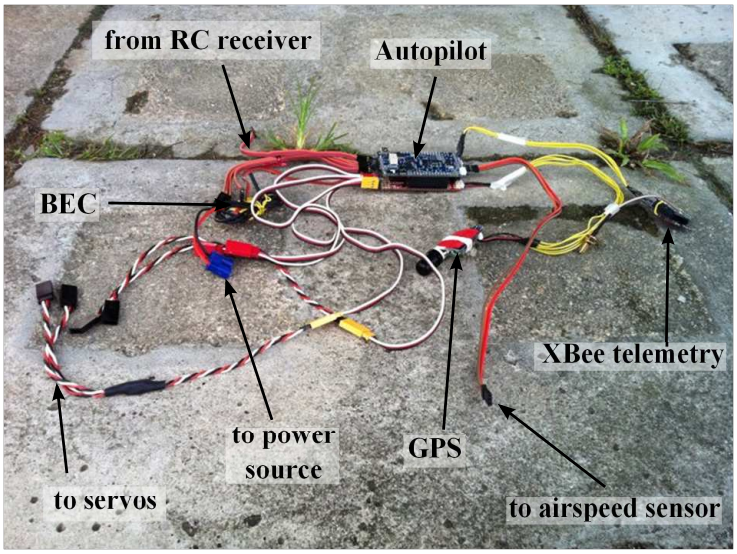


Figure 10.9: Ardupilot Mega with various connections

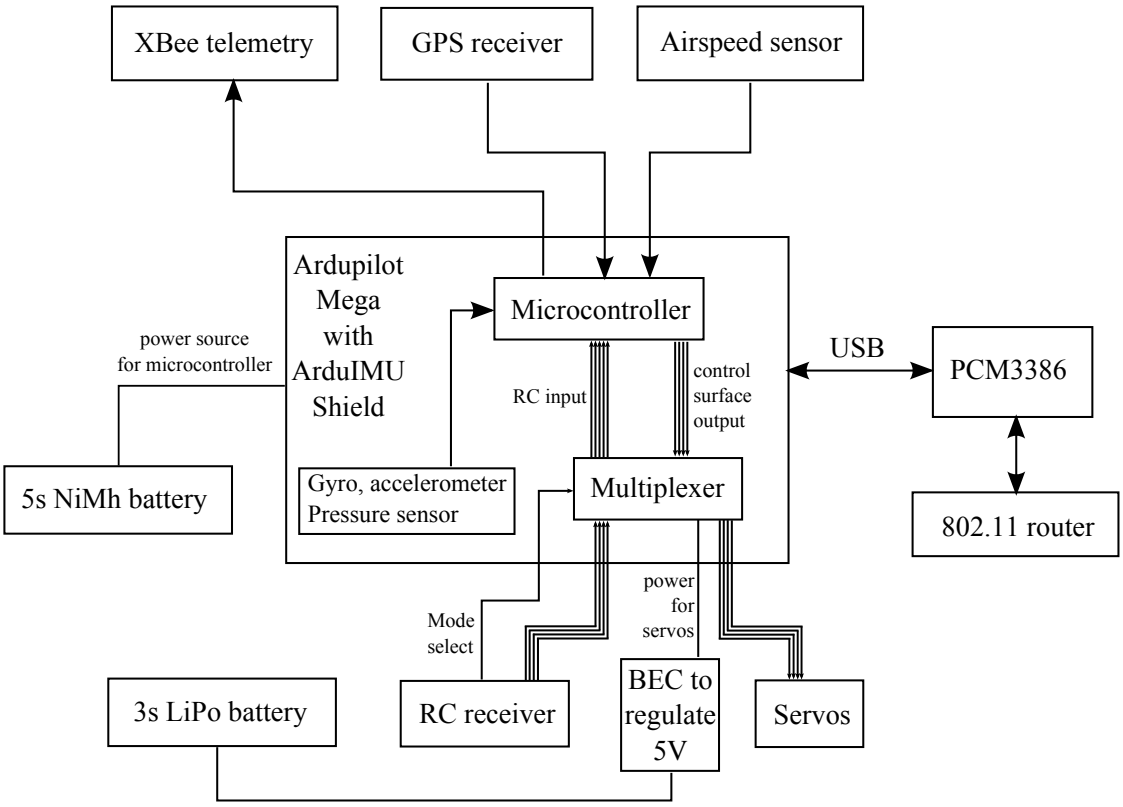


Figure 10.10: Overall system architecture

10.1.6 Overall system architecture

The overall system architecture is presented in Figure 10.10. The Ardupilot Mega connected to the various sensors and other connecting wires is shown in Figure 10.9.

10.2 Experiments for data communication

We use the hardware setup described in Section 10.1 on the 1/150 scale Pilatus PC-6 Porter Scale 150 to test the viability of air-to-ground and air-to-air communication. The PID based control system was used on the Ardupilot Mega. Two UAVs were flown to act as relays between the base station and another distant target point on the ground. The aim was to collect data from a data source placed at the target point and relay it back to the base station using store-carry-forward routing on the UAVs. Experimental results showed a Packet Delivery Rate of 100% at a bitrate of over 1Mbps.

10.3 Experiments for control system

We use the hardware setup described in Section 10.1 on the Multiplex Mentor to test and validate the effectiveness of the control system described in Chapter 8. The system architecture used was the same as the one illustrated in Figure 10.10, except without the PCM3386 and the router. Using this setup, we conducted two sets of experiments:

1. To test controller performance in straight line tracking (i.e. waypoint navigation)

2. To test controller performance in circular trajectory tracking (for PR state behavior [refer to Section 9.4.4])

For both sets of experiments, we conducted Hardware-in-the-loop (HWIL) tests as well as real flight tests. HWIL tests were performed to conduct controlled experiments where the controller's performance could be tested under different wind scenarios. One of the disadvantages with field tests is the inability to measure wind speed and direction accurately, let alone the inability to control wind speeds. In order to study the performance of the controller as implemented on actual hardware, but under controlled environmental settings, we use the Ardupilot Mega to control an aircraft within the X-Plane 9 simulator. In HWIL simulation, the controller (Ardupilot Mega) receives emulated sensor readings from the X-Plane 9 simulator, generates control decisions, and sends control commands back to the simulator, which then simulates the behavior of the aircraft. The setup is achieved through the use of ArduSim [92], which is a software that acts as a bridge between X-Plane and Ardupilot. The setup for HWIL tests is shown in Figure 10.11.

10.3.1 Straight line tracking

The controller used for straight line tracking is the exact same as the one described in Chapter 8. The DCS was trained using data from simulation using the default PID based controller on the Ardupilot Mega, tuned separately for 3 different fixed wind speeds - 0, 10, and 20kts. The DCS training constants were set such that the resulting DCS network had no more than 40 neurons. The reason for this was the limited memory (8K) on the Atmega1280. We tested the memory capacity by adding arbitrary neurons to verify the point of overflow. With any more than 40 neurons, the Atmega1280 ended up with a memory overflow. The final set of DCS

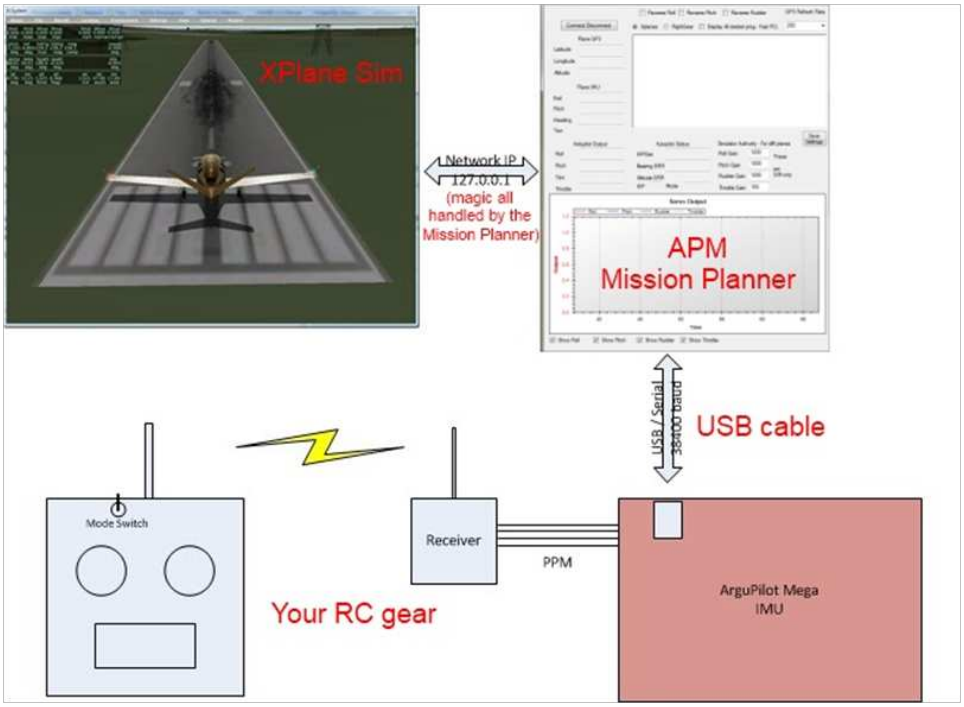


Figure 10.11: HWIL setup

constants used were:

Kohonen constants,	ε_{bmu}	=	0.08
	ε_{Nh}	=	0.012
Connection decay,	γ	=	0.952
Connection threshold,	θ	=	0.01
Rate of gradient descent in output,	η	=	0.08

The resulting DCS network consisted of 34 neurons. The number of neurons in the learned network is a result of neuron additions, interactions and deletions through hundreds of iterations. Getting this number to an exact 40 would have been near to impossible. As a result, we settle with the DCS network consisting of 34 neurons. The average absolute error achieved in terms of commanded roll angle when the DCS was tested against the training data was 4.22 degrees (in roll angle command).

Table 10.1: Comparison of average absolute cross-track error for the two controllers under different wind conditions

Wind Direction (Degrees)	Wind Speed (Knots)	Average cross-track error (PID) (ft)	Average cross-track error (DCS) (ft)	Improvement (%)
100°, switch to 300° at t=200s	1 average	11.06	11.00	0.59
100°, switch to 300° at t=200s	10 average	30.38	21.52	29.16
100°, switch to 300° at t=200s	20 average	97.34	57.94	40.48
$0° - 359°$ sinusoidal	15 average	81.81	51.17	37.45

Results from HWIL simulation

The HWIL simulations were first performed with the default PID based controller on the Ardupilot Mega followed by the DCS based controller for 4 different wind speed and direction settings. Table 10.1 shows the various wind conditions and results obtained. The results show a consistent improvement in performance by the DCS as compared to the PID based controller. The percentage improvement is higher at higher wind speeds, which can be attributed to the fact that the default PID controller on the Ardupilot is unable to correct for crosswinds.

Results from real flight tests

The tests were conducted over a 2 hour period between 7.20 am and 9.20 am, on a very light-weight foam-built Multiplex Mentor electric RC airplane. The flight timings were constrained by air traffic rules to the early morning period. The weather was clear, and as is typical of early morning weather in Singapore on a

clear day, winds started off fairly still at the beginning and picked up in intensity as the sun rose. As it is not possible to measure the wind speed and direction at cruise altitude (and even less so to control them), three runs of two tests (one with default Ardupilot PID and one with DCS) were conducted with each test lasting about 10 minutes (i.e. 20 minutes per run) to mitigate the varying wind conditions. The PID and DCS tests were also run alternately to further minimize differences. The pattern flown by the aircraft is illustrated in Figures 10.12-10.14. The aircraft was required to follow a line segment of length 1000ft. At the end of the line segment on either side, the aircraft was required to circle around and follow the line segment in the opposite direction. Experimental results are presented below.

1. First run:

   The first run took place in moderately calm winds at an altitude of 130m. The PID showed an average deviation from the line segment of 29.61 feet, while the DCS showed an average deviation of 23.76 feet, an improvement of 19.76%. Figure 10.12 shows the telemetry plot obtained from the Ardupilot Megas onboard logs. The DCS flight path is shown in yellow while the PID flight path is shown in blue.

2. During the second run, the wind conditions had not changed much. Conditions were still calm. The PID showed an average deviation of 24.54 feet while the DCS showed a deviation of 24.10 feet, an improvement of 1.79%. Figure 10.13 shows the corresponding telemetry plot.

3. Winds started to pick up at about 8.45 am, and were fairly strong by the third tests conducted at 8.55 am (PID) and 9.10 am (DCS). The PID controller had an average deviation of 41.04 feet while the DCS had an average deviation of 29.54 feet, an improvement of 28.02%. Figure 10.19b shows the corresponding telemetry plot.

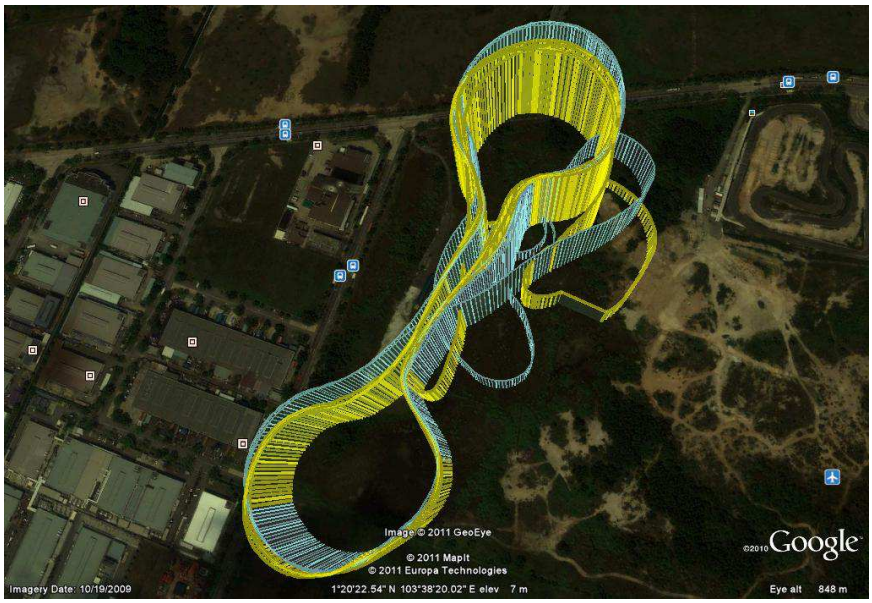


Figure 10.12: Telemetry plot of flight paths with PID (blue) and DCS (yellow) based controllers for first run in straight line tracking tests

Figure 10.13: Telemetry plot of flight paths with PID (blue) and DCS (yellow) based controllers for second run in straight line tracking tests

Figure 10.14: Telemetry plot of flight paths with PID (blue) and DCS (yellow) based controllers for third run in straight line tracking tests

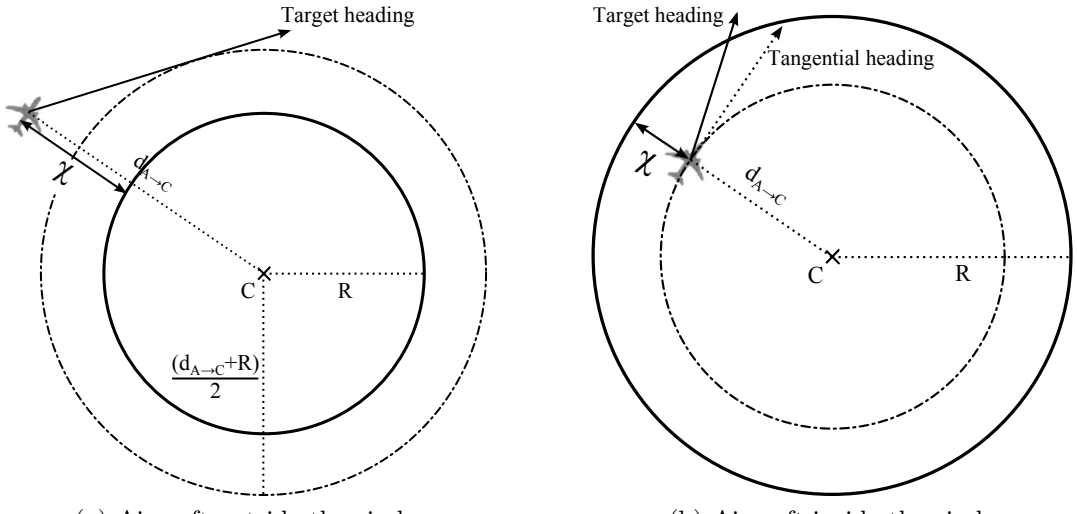
Table 10.2: Comparison of average absolute cross-track error achieved by the two controllers for the 3 straight line tracking tests

	Average cross-track error (PID) (ft)	Average cross-track error (DCS) (ft)	Improvement (%)
First run	29.61	23.76	19.76
Second run	24.54	24.10	1.79
Third run	41.04	29.54	28.02

The consolidated results of the real flight tests are shown in Table 10.2. We notice that under calm winds, the DCS based controller provides little improvement over the PID based controller. However with winds in effect, the PID deteriorates substantially whereas the DCS manages to maintain a low cross-track error, thus giving an improved performance by 28%. Given that a distance of 1000ft is actually short for an aircraft moving at 15m/s, thus giving it very little time to settle in to the line segment, we believe the deviation results are an indication of successful crosswind correction by the DCS.

10.3.2 Circular trajectory tracking

The DCS based controller used for circular trajectory tracking was the same as that described in Chapter 8. However, the controller used to train the DCS was different owing to the difference in required behavior. The behavior of the PID based controller used to train the DCS is shown in Figures 10.15a and 10.15b.



(a) Aircraft outside the circle

(b) Aircraft inside the circle

Figure 10.15: Control mechanism of PID based controller for circular trajectory tracking

The function of the controller is to enable navigation of the UAV along a required circle of radius, r, centered at a point C, whose latitude and longitude are given

by λ_C and ϕ_C. Let us also define λ_A and ϕ_A to be the latitude and longitude of the current position of the aircraft, A. We then obtain the cross-track distance, χ, as follows (standard distance formula using GPS coordinates [93]):

$$d_{A\to C} = \arccos(\ \sin\lambda_C \sin\lambda_A + \cos\lambda_C \cos\lambda_A \cos(\phi_C - \phi_A))$$

$$\chi = d_{A\to C} - R$$

The meaning of χ is illustrated in Figures 10.15a and 10.15b. We use this above computed cross-track distance, χ, as the parameter for lateral control. The aim of the controller is now to asymptotically minimize χ to 0.

The controller determines and achieves the target course. Note that target course refers to the direction in which the aircraft GPS course should head and not necessarily the direction in which the aircraft's nose points. When the aircraft is outside the desired circle, the aircraft aims to fly along the tangent connecting it to a circle of radius $\frac{d_{A\to C}+R}{2}$. The result is that aircraft converges to the desired circle with an exponential decay where the error drops faster initially and converges smoothly. When the aircraft is inside the circle, the target course is given by a PID component that uses χ, the cross-track error, to produce the target course. The target course is then achieved by the underlying dual-loop PID controller. The structure of the PID based controller is shown in Figure 10.16. The shaded region in Figure 10.16 represents the portion of the controller that is wind dependent and that is what is replaced by the DCS after training.

The PID gains for the wind dependent outer loop components (i.e. shaded region in Figure 10.16) are tuned separately for 3 different fixed wind speed situations. Three sets of gains are generated for each of the wind speeds of 0, 10, and 20kts. The final data set consisting of 4500 inputs is then used to train the DCS through

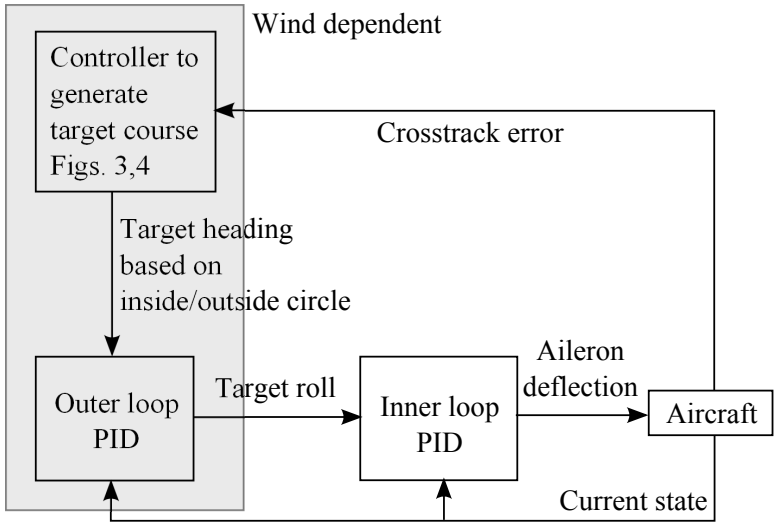


Figure 10.16: Block diagram of PID based controller

supervised learning. For training purposes, the DCS constants are set as follows:

Kohonen constants,	ε_{bmu}	=	0.08
	ε_{Nh}	=	0.012
Connection decay,	γ	=	0.952
Connection threshold,	θ	=	0.01
Rate of gradient descent in output,	η	=	0.08

The number of neurons in the network at the end of training is 37. The average absolute error achieved when the DCS is tested against the training data is 3.47 degrees (in roll angle command).

Results from HWIL simulation

The HWIL simulations are first performed with the default PID based controller on the Ardupilot Mega followed by the DCS based controller for 4 different wind speed and direction settings. Table 10.3 shows the various wind conditions and results obtained. The results show a consistent improvement of between 37.28%

and 60.58%.

Table 10.3: Comparison of average absolute cross-track error for the two controllers under different wind conditions

Wind Direction (Degrees)	Wind Speed (Knots)	Average cross-track error (PID) (ft)	Average cross-track error (DCS) (ft)	Improvement (%)
100°, switch to 300° at t=200s	1 average	2.79	1.75	37.28
100°, switch to 300° at t=200s	10 average	48.35	19.06	60.58
100°, switch to 300° at t=200s	20 average	90.51	55.71	38.45
$0° - 359°$ sinusoidal	15 average	50.41	25.02	50.37

Results from real flight tests

The tests were conducted over a 1.5 hour period between 8.05 am and 9.35 am, on a very light-weight foam-built Multiplex Mentor electric RC airplane. The flight timings were constrained by air traffic rules to the early morning period. The weather was clear, and as is typical of early morning weather in Singapore on a clear day, winds started off fairly still at the beginning and picked up in intensity as the sun rose. As it is not possible to measure the wind speed and direction at cruise altitude (and even less so to control them), three runs of two tests (one with default Ardupilot PID and one with DCS) were conducted with each test lasting about 6 minutes (i.e. 12 minutes per run) to mitigate the varying wind conditions. The PID and DCS tests were also run alternately to further minimize differences.

1. First run:

Table 10.4: Comparison of average absolute cross-track error achieved by the two controllers for the 3 flight tests

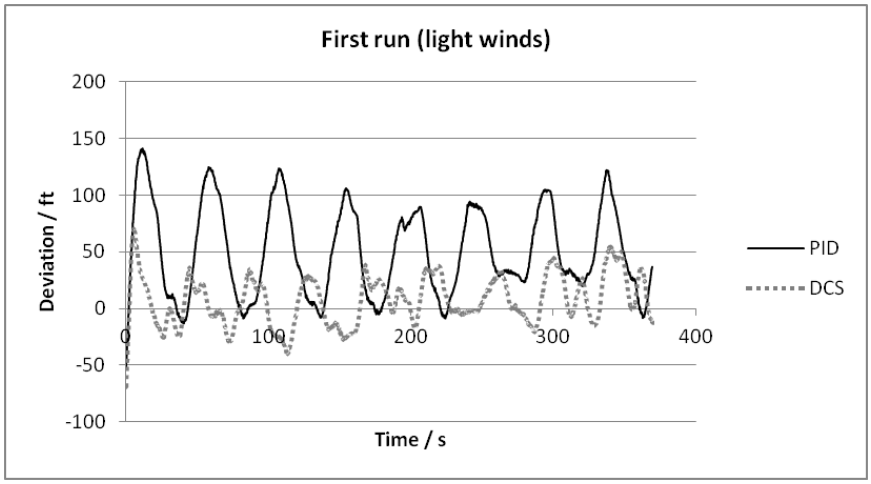
	Average cross-track error (PID) (ft)	Average cross-track error (DCS) (ft)	Improvement (%)
First run	51.7	18.5	64.2
Second run	73.8	36.8	50.1
Third run	80.8	33.9	58.0

The first run took place in moderately calm winds at an altitude of 50m. The PID showed an average deviation from the circle of 51.7 feet, while the DCS showed an average deviation of 18.5 feet, an improvement of 64.2%. Figure 10.17a shows a plot of cross-track error against time for the first run. Figure 10.17b shows the telemetry plot obtained from the Ardupilot Megas onboard logs. The DCS flight path is shown in yellow while the PID flight path is shown in blue.

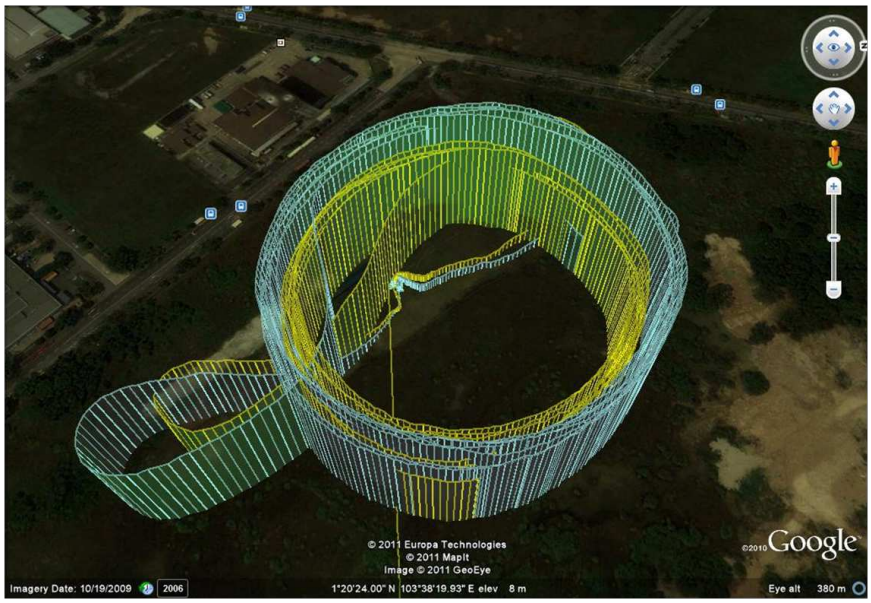
2. The second run took place in moderate winds. The PID showed an average deviation of 73.8 feet while the DCS showed a deviation of 36.8 feet, an improvement of 50.1%. Figure 10.18a shows a plot of cross-track error against time for the second run. Figure 10.18b shows the corresponding telemetry plot.

3. Winds started to pick up at about 8.55 am, and were fairly strong by the third tests conducted at 9.07 am (PID) and 9.20 am (DCS). The PID controller had an average deviation of 80.8 feet while the DCS had an average deviation of 33.9 feet, an improvement of 58%. Figure 10.19a shows a plot of cross-track error against time for the second run. Figure 10.19b shows the corresponding telemetry plot.

The consolidated results of the real flight tests are shown in Table 10.4. In all three runs the DCS improved performance by between 50 and 64%, validating the lab test

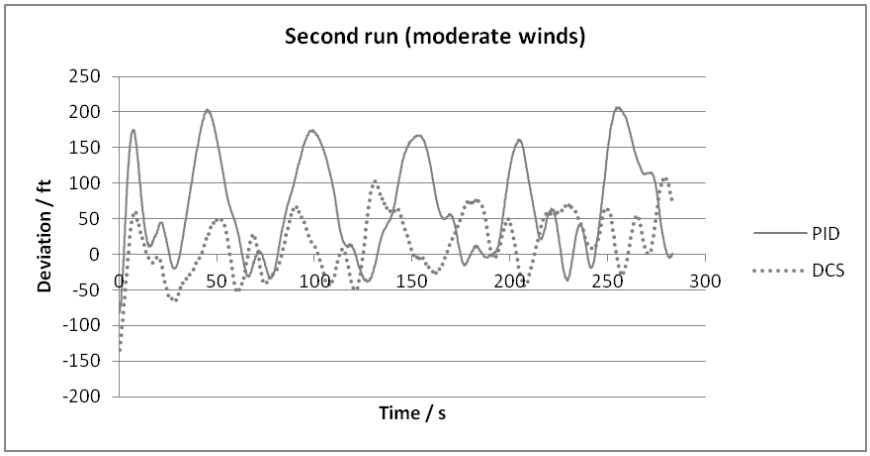


(a) Cross-track error against time

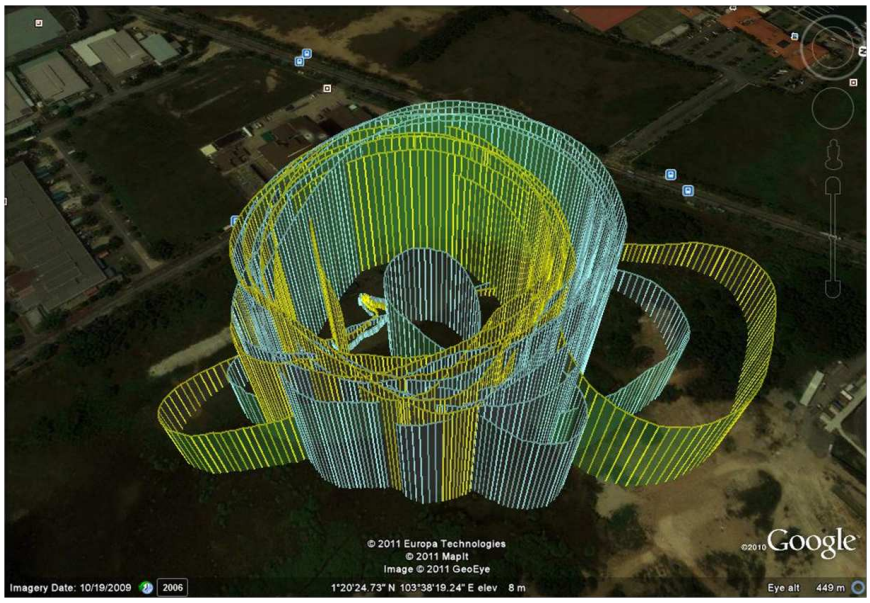


(b) Telemetry plot of flight paths with PID (blue) and DCS (yellow) based controllers

Figure 10.17: Results from first run for circular trajectory tracking

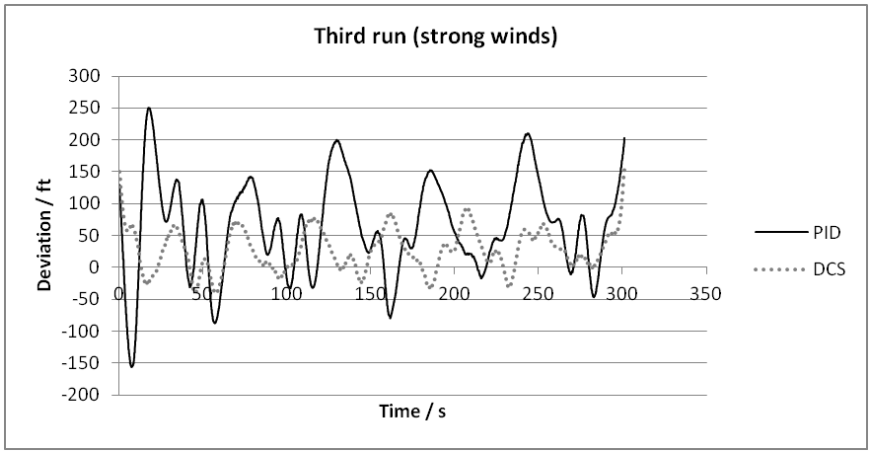


(a) Cross-track error against time



(b) Telemetry plot of flight paths with PID (blue) and DCS (yellow) based controllers

Figure 10.18: Results from second run for circular trajectory tracking



(a) Cross-track error against time

(b) Telemetry plot of flight paths with PID (blue) and DCS (yellow) based controllers

Figure 10.19: Results from third run for circular trajectory tracking

results and that the DCS greatly improves tracking performance in cross-winds. The deviations are fairly large (up to 80 feet for the PID controlled autopilot and 37 feet for the DCS controlled autopilot), due to the very light Multiplex Mentor foam platform that was used.

10.4 Summary and Contributions

In this chapter we detailed the field tests and HWIL tests that were performed to validate the control system behavior and prove the viability of air-ground and air-air communication. Details of the equipment and platforms used were laid out for reference and for easy future deployment. We showed that the DCS implemented on an Ardupilot Mega platform managed to successfully correct for crosswind effects and maneuver the aircraft along the desired path. Accurate control and ability to communicate are the basic building blocks for the entire control and coordination mechanism proposed in Chapters 7-9. We believe that these are two components that are affected the most when moved from simulation to reality. Having shown the practical viability of both, simulation of the higher level mechanisms makes it sufficient to show the feasibility of the entire system.

The main contribution of this chapter is the deployment ready implementation of the proposed algorithms. Results from practical field tests serve to show the applicability and viability of everything else that has been discussed in this thesis.

Chapter 11

Conclusion

11.1 Summary of the Thesis

In this thesis, we addressed the problem of using UAVs to autonomously build a wireless communications backbone. We studied the problem from a theoretical perspective with the aim of determining paths for M agents so as to minimize the worst case latency in a DTN of N ground nodes. To this end, we proposed the Bounded-Edge Count Diametric Latency Minimizing Steiner Tree (BECDLMST) as the solution for the agent path design problem with complete *a priori* information. Finding the optimal BECDLMST was proven to be NP-hard. Subsequently an exact exponential algorithm for BECDLMST was presented, that was designed to prune the solution space as much as possible at every step so as to minimize computation time. Experiments showed that the algorithm was capable of handling up to 30 ground stations and 40 agents. Comparisons were also made against FRA, which provides the better of performances among existing methods. The results showed a considerable improvement in maximum network latency achieved by BECDLMST as compared to FRA. Apart from providing a centralized solution for the case with perfect information, the theoretical analysis provided insight into

designing decision mechanisms in a decentralized situation. Noting the exponential complexity of the exact algorithm, we proposed an anytime heuristic to generate near-optimal BECDLMSTs. The heuristic used update rules and iterative tree evolution strategies that enabled the self-organization of rendezvous points. The algorithm ensured that at the end of every iteration, the tree satisfied a valid solution, thus allowing anytime termination. Empirical results were used to prove that the algorithm was capable of handling very large data sets in terms of ground nodes and agents. The heuristic brought the computation complexity from an exponential one down to a quadratic one, thereby making BECDLMST a feasible solution to the agent path design problem.

Following the theoretical analysis, we embarked on the combined problem of coverage, search and tethering under realistic winds and communication limitations. We proposed a decentralized, hierarchical architecture for control and coordination. The control component of the system looked into the issue of wind effects, specifically crosswinds that deflect lightweight UAVs from their intended paths. A DCS-NDI controller capable of correcting for such crosswind effects was presented. We showed how the traditional heading parameter is unsuitable in the presence of winds and introduce the use of cross-track error, χ, as the control parameter. The DCS-based approach was shown to be capable of controlling the nonlinear system, by learning the behaviors of individually tuned PID-controllers and interpolating between them to achieve accurate waypoint navigation. The DCS in particular was modified to learn more accurately and faster as compared to the original DCS (specifically a 10 times better representation in about $\frac{1}{6}$th the time for the dataset we used). Simulation results showed the controller achieved a maximum deflection of $\approx$ 10ft from the intended flight path in the presence of winds with speeds up to 30kts. The controller thus proved the viability of a reactive system for crosswind correction without the need for wind speed and direction measurements. It also al-

lowed more flexibility at the upper coordination layer by letting it not consider the adverse effects of wind. At the higher layer, for the multiagent coordination problem, a "near-decision-theoretic" approach was taken in which an agent's decision depended on maximizing a scoring function. The uncertainty and complete lack of *a priori* information called for an adaptive solution, which in our case was achieved through the presented adaptive finite state machine. The belief information used within the operational states, was chosen such that the same information could be used for search as well as relay state decisions, thus enabling a hybrid search and relay state. The behavior of agents in the relay state was designed to produce an emergent chain relay architecture, in an effort to mimic the BECDLMST solution structure derived using theoretical analysis. Comparisons were also made between network latencies achieved by the decentralized solution with relay state agents and the centralized heuristic, to show that decentralizing the solution only resulted in a 10-50% increase in latency. An effective neural-network-based belief information exchange mechanism was proposed in order to use minimal bandwidth, thus allowing ground station related data transmission to utilize higher bandwidth. The novel coordination mechanism thus proposed, was empirically shown to be capable of achieving a non-monotonic increase in Q, which is a metric that increases with higher cell visit frequency and lower packet latency. Results also showed the resilience of the coordination algorithm to changing situations such as addition or loss of ground stations.

Finally, we provided results from real flight tests conducted using the proposed controller. The effectiveness of the controller and the viability of air-to-ground and air-to-air communication were proven. Although the number of flight tests performed was of a small order, each test experienced different wind conditions. Within each test, the straight line or circular trajectory was repeatedly followed a number of times thus making each test a collection of a number of samples. The

varying wind conditions with sun rise on the other hand, accounted for a reasonable spread in test conditions, thus making it representative of normal Singapore weather. The results obtained were therefore statistically representative of control behavior in general Singapore weather.

11.2 Future work

Future research in this area could explore the use of rigid UAV formations to represent a single agent, resulting in agents with much larger sensing areas. The trade-off between assigning a UAV to a formation and treating it as an individual agent could provide further insight into the problem.

As opposed to only fixed-wing UAVs, future research could explore the use of a combination of rotary wing aircrafts and fixed wing aircrafts. The benefit of rotary wing aircrafts is their high maneuverability and their ability to hover, whereas fixed wing aircrafts can achieve higher speeds. A heterogeneous set of aircrafts could utilize the benefits of both. The problem would then likely involve a component that is similar to the problem of sensor placement.

This thesis considered areas that were fully accessible without any obstacles. However, the presence of obstacles is likely in areas with tall buildings. Future research could explore the same problem with inaccessible regions in a given area and study how that would affect both, the centralized solution as well as the decentralized one.

Bibliography

[1] A. Hafslund, T. T. Hoang, and O. Kure, "Push-to-talk applications in mobile ad hoc networks," in *IEEE Vehicular Technology Conference.*, pp. 2410–2414, 2005. 1

[2] S. Masaki and C. Wataru, "Helicopter satellite communication system for disaster control operations ensuring prompt communications," *Journal of the IEICE*, vol. 89, no. 9, pp. 806–810. 1

[3] Iridium, "Iridium," 2009. `http://www.iridium.com/`. 1

[4] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing* (T. Imielinski and H. F. Korth, eds.), vol. 353 of *The Springer International Series in Engineering and Computer Science*, pp. 153–181, Springer US, 1996. 2

[5] Y.-B. Ko and N. H. Vaidya, "Location-aided routing (lar) in mobile ad hoc networks," *Wireless Networks*, vol. 6, pp. 307–321, July 2000. 2

[6] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture." RFC 4838, 2007. 3, 128

[7] M. Khabbaz, C. Assi, and W. Fawaz, "Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges," *Communications Surveys Tutorials, IEEE*, vol. PP, no. 99, pp. 1 –34, 2011. 3

[8] F. Bai and A. Helmy, "Impact of mobility on mobility assisted information diffusion (maid) protocols," tech. rep., 2005. 3

[9] Y. Chen, K. Moore, and Z. Song, "Diffusion boundary determination and zone control via mobile actuator-sensor networks (mas-net) challenges and opportunities," in *SPIE Conference on Intelligent Computing: Theory and Applications II*, 2004. 4

[10] K. Moore and Y. Chen, "Model-based approach to characterization of diffusion processes via distributed control of actuated sensor networks," in *IFAC Symposium of Telematics Applications in Automation and Robotics*, 2004. 4

[11] J. Jennings, G. Whelan, W. Evans, and , "Cooperative search and rescue with a team of mobile robots," in *ICAR '97. Proceedings., 8th International Conference on Advanced Robotics*, pp. 193–200, 1997. 4

[12] J. Schlecht, K. Altenburg, B. M. Ahmed, and K. E. Nygard, "Decentralized search by unmanned air vehicles using local communication," in *Proc. International Conference on Artificial Intelligence*, (Las Vegas, NV), pp. 757–762, 2003. 4

[13] B. Argrow, D. Lawrence, and E. Rasmussen, "UAV systems for sensor dispersal, telemetry, and visualization in hazardous environments," in *43rd Aerospace Sciences Meeting and Exhibit*, 2005. 4

[14] J. Maslanik, J. Curry, S. Drobot, and G. Holland, "Observations of sea ice using a low-cost unpiloted aerial vehicle," in *16th IAHR International Symposium on Sea Ice*, 2002. 4

[15] F. R. Noreils, "Multi-robot coordination for battlefield strategies," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Raleigh, NC), 1992. 4

[16] K. Xu, X. Hong, M. Gerla, *et al.*, "Landmark routing in large wireless battlefield networks using UAVs," in *IEEE MILCOM '01*, 2001. 4

[17] D. Hague, H. T. Kung, and B. Suter, "Field experimentation of cots-based UAV networking," in *Military Comm. Conference.*, pp. 1–7, 2006. 4

[18] B. Towles and W. J. Dally, "Worst-case traffic for oblivious routing functions," *IEEE Computer Architecture Letters*, vol. 1, pp. 1–8, 2002. 12

[19] W. Peng, B. Zhao, W. Yu, C. Wu, and X. Yan, "Ferry route design with delay bounds in delay-tolerant networks," in *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*, pp. 281 –288, 29 2010-july 1 2010. 15

[20] T. Khalaf and S. W. Kim, "Delay analysis in message ferrying system," in *IEEE International Conference on Electro/Information Technology, 2008. EIT 2008.*, pp. 333 –336, may 2008. 15

[21] V. Kavitha and E. Altman, "Queuing in space: Design of message ferry routes in static ad hoc networks," in *Teletraffic Congress, 2009. ITC 21 2009. 21st International*, pp. 1 –8, sept. 2009. 15

[22] A. Kansal, A. A. Somasundara, D. D. Jea, M. B. Srivastava, and D. Estrin, "Intelligent fluid infrastructure for embedded networks," in *Proc. MobiSys '04*, pp. 111–124, 2004. 15

[23] W. Zhao, M. Ammar, and E. Zegura, "Controlling the mobility of multiple data transport ferries in a delay-tolerant network," in *Proc. IEEE INFOCOM'05.*, vol. 2, pp. 1407 – 1418 vol. 2, mar. 2005. 15, 16, 17, 18, 19, 54, 56, 57, 75, 76, 116

[24] K. Almi'ani, A. Viglas, and L. Libman, "Mobile element path planning for time-constrained data gathering in wireless sensor networks," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pp. 843 –850, april 2010. 15

[25] A. Somasundara, A. Ramamoorthy, and M. Srivastava, "Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pp. 296 – 305, dec. 2004. 15

[26] R. Yu, X. Wang, and S. Das, "Efficient data gathering using mobile elements in partially connected sensor networks," in *Control and Decision Conference, 2008. CCDC 2008. Chinese*, pp. 5337 –5342, july 2008. 15

[27] J. Wang, "Trajectory optimization of mobile element for sparse wireless sensor networks," in *International Conference on Communication Software and Networks, 2009. ICCSN '09.*, pp. 829 –833, feb. 2009. 15

[28] K. Gandhi and P. Narayanasamy, "Energy- aware mobile element scheduling in wireless sensor networks," in *3rd International Conference on Sensing Technology, 2008. ICST 2008.*, pp. 107 –113, 30 2008-dec. 3 2008. 15

[29] H. Miura, D. Nishi, N. Matsuda, and H. Taki, "Message ferry route design based on clustering for sparse ad hoc networks," in *Knowledge-Based and Intelligent Information and Engineering Systems* (R. Setchi, I. Jordanov, R. Howlett, and L. Jain, eds.), vol. 6277 of *Lecture Notes in Computer Science*, pp. 637–644, Springer Berlin / Heidelberg, 2010. 15

[30] Z. Zhang and Z. Fei, "Route design for multiple ferries in delay tolerant networks," in *IEEE WCNC 2007*, pp. 3460 –3465, mar. 2007. 15, 16, 17, 116

[31] B. Jedari, R. Goudarzi, and M. Dehghan, "Efficient routing using partitive clustering algorithms in ferry-based delay tolerant networks," *International Journal of Future Generation Communication and Networking*, vol. 2, 2009. 16, 17, 116

[32] J. Wu, S. Yang, and F. Dai, "Logarithmic store-carry-forward routing in mobile ad hoc networks," *Parallel and Distributed Systems, IEEE Trans.*, vol. 18, pp. 735 –748, jun. 2007. 18, 19, 116

[33] P. Crescenzi, V. Kann, M. Halldrsson, and M. Karpinski, "Minimum geometric steiner tree," in *Approximation on the web: A compendium of NP optimization problems* (J. Rolim, ed.), vol. 1269 of *Lecture Notes in Computer Science*, pp. 111–118, Springer Berlin / Heidelberg, 1997. 21

[34] J. M. Ho and D. T. Lee, "Bounded diameter minimum spanning trees and related problems," in *Proceedings of the fifth annual symposium on Computational geometry*, SCG '89, pp. 276–282, 1989. 27, 28, 30

[35] H. W. E. Jung, "ber den kleinsten kreis, der eine ebene figur einschliesst," *J. reine angew. Math.*, vol. 137, pp. 310–313, 1910. 28

[36] L. D. Drager, J. M. Lee, and C. F. Martin, "On the geometry of the smallest circle enclosing a finite set of points," *Journal of the Franklin Institute*, vol. 344, no. 7, pp. 929 – 940, 2007. 53

[37] C. R. Reeves, ed., *Modern heuristic techniques for combinatorial problems*. New York, NY, USA: John Wiley & Sons, Inc., 1993. 60

[38] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989. 61

[39] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996. 61

[40] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pp. 39 –43, oct 1995. 63

[41] T. X. Brown, B. M. Argrow, C. Dixon, S. Doshi, R.-G. Thekkekunnel, and D. Henkel, "Ad hoc UAV ground network (AUGNet)," in *AIAA's 3rd Unmanned Unlimited Technical Conference*, no. AIAA-2004-6321, (Chicago, IL), 20-23 Sep. 2004. 84

[42] S. Rathinam, M. Zennaro, T. Mak, and R. Sengupta, "An architecture for UAV team control," in *5th IFAC Symposium on Intelligent Autonomous Vehicles*, 2004. 84

[43] G. Vachtsevanos, L. Tang, and J. Reimann, "An intelligent approach to coordinated control of multiple unmanned aerial vehicles," in *American Helicopter Society 60th Annual Forum*, 2004. 84

[44] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 3, pp. 1936 – 1941 vol.3, 2002. 85

[45] A. Richards, T. Bellingham, and J. How, "Coordination and control of multiple UAVs," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference, Monterey, CA*, Aug. 2002. 85

[46] C. Schumacher, M. Pachter, P. Chandler, and L. Pachter, "UAV task assignment with timing constraints via mixed-integer linear programming," in *Proceedings of the AIAA 3rd Unmanned Unlimited Systems Conference*, 2004. 85

[47] C. Schumacher, P. Chandler, M. Pachter, and L. Pachter, "Optimization of air vehicles operations using mixed integer linear programming," *Journal of the Operations Research Society*, vol. 58, no. 4, pp. 516–527, 2007. 85

[48] S. Rasmussen, T. Shima, and S. J. Rasmussen, *UAV Cooperative Decision and Control: Challenges and Practical Approaches*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008. 85

[49] T. Shima, P. Chandler, and M. Pachter, "Decentralized estimation for cooperative phantom track generation," in *Cooperative Systems*, vol. 588 of *Lecture Notes in Economics and Mathematical Systems*, pp. 339–350, Springer Berlin Heidelberg, 2007. 85

[50] D. S. Bernstein, S. Zilberstein, and N. Immerman, "The complexity of decentralized control of markov decision processes," in *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, (San Francisco, CA, USA), pp. 32–37, Morgan Kaufmann Publishers Inc., 2000. 85

[51] M. P. Wellman and P. R. Wurman, "Market-aware agents for a multiagent world," *Robotics and Autonomous Systems*, vol. 24, pp. 115–125, 1997. 85

[52] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "An asynchronous complete method for general distributed constraint optimization," in *In Proc of Autonomous*

Agents and Multi-Agent Systems Workshop on Distributed Constraint Reasoning, 2002. 85

[53] D. T. Cole, A. H. Gktoan, and S. Sukkarieh, "The demonstration of a cooperative control architecture for UAV teams," in *Experimental Robotics*, vol. 39 of *Springer Tracts in Advanced Robotics*, pp. 501–510, Springer Berlin / Heidelberg, 2008. 85, 86

[54] G. J. Ducard, "Nonlinear aircraft model," in *Fault-tolerant Flight Control and Guidance Systems*, Advances in Industrial Control, pp. 27–41, Springer London, 2009. 86, 110

[55] B. Yun, B. M. Chen, K. Y. Lum, and T. H. Lee, "Design and implementation of a leader-follower cooperative control system for unmanned helicopters," *Journal of Control Theory and Applications*, vol. 8, pp. 61–68, February 2010. 86, 110

[56] J. How, Y. Kuwata, and E. King, "Flight demonstrations of cooperative control for UAV teams," in *Proceedings of the 3rd Conference Unmanned Unlimited Technical Conference, Chicago*, Sept. 2004. 86, 110

[57] D. Allerton, *Principles of Flight Simulation.* Wiley Publishing, 2009. 92

[58] C. G. Kllstrm, "Autopilot and track-keeping algorithms for high-speed craft," *Control Engineering Practice*, vol. 8, no. 2, pp. 185 – 190, 2000. 92

[59] R. S. Christiansen, "Design of an autopilot for small unmanned aerial vehicles," Master's thesis, 2004. Adv.-Beard, Randy. 92

[60] NASA, "Radio navigation," 2010. `http://virtualskies.arc.nasa.gov/navigation/3.html`. 92

[61] C. Tournes and B. Landrum, "Adaptive control of aircraft lateral directional axis using subspace stabilization," in *System Theory, 2001. Proceedings of the 33rd Southeastern Symposium on*, pp. 425 –429, mar 2001. 92

[62] D. Doman and M. Oppenheimer, "Improving control allocation accuracy for nonlinear aircraft dynamics," in *Proceedings of AIAA Guidance, Navigation, and Control Confrence, Monterey, CA*, Aug. 2002. 93

[63] M. G. Perhinschi, M. L, G. Campa, M. R. Napolitano, and M. L. Fravolini, "Online parameter estimation issues for the nasa ifcs f-15 fault tolerant systems," in *in Proceedings of the American Control Conference*, pp. 8–10, 2002. 96

[64] D. Ito, J. Georgie, J. Valasek, and D. T. Ward, "Reentry vehicle flight controls design guidelines: Dynamic inversion," Tech. Rep. NASA/TP-2002-210771, 2002. 96, 97

[65] A. Isidori, *Nonlinear Control Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1995. 96

[66] J. Bruske and G. Sommer, "Dynamic cell structure learns perfectly topology preserving map," *Neural Comput.*, vol. 7, no. 4, pp. 845–865, 1995. 98, 99

[67] C. C. Jorgensen, "Direct adaptive aircraft control using dynamic cell structure neural networks," in *NASA TM 112198*, 1997. 98

[68] T. Martinetz, "Competitive hebbian learning rule forms perfectly topology preserving maps," in *Proc. ICANN'93, Int. Conference on Artificial Neural Networks* (S. Gielen and B. Kappen, eds.), (London, UK), pp. 427–434, Springer, 1993. 101

[69] M. Batalin and G. Sukhatme, "The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005.*, pp. 3478 – 3485, 18-22 2005. 114

[70] C. S. Kong, N. A. Peng, and I. Rekleitis, "Distributed coverage with multi-robot system," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 2423 –2429, 15-19 2006. 114

[71] S. S. Ge and C. Fua, "Complete multi-robot coverage of unknown environments with minimum repeated coverage," in *Proceedings of the 2005 IEEE International*

Conference on Robotics and Automation, 2005. ICRA 2005., pp. 715 – 720, 18-22 2005. 114

[72] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset, "Limited communication, multi-robot team based coverage," in *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04.*, vol. 4, pp. 3462 – 3468 Vol.4, april 2004. 114

[73] A. Krause and C. Guestrin, "Near-optimal nonmyopic value of information in graphical models," in *21st Conference on Uncertainty in Artificial Intelligence (UAI*, p. 5, 2005. 115

[74] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg, "Near-optimal sensor placements: maximizing information while minimizing communication cost," in *IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks*, (New York, NY, USA), pp. 2–10, ACM, 2006. 115

[75] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, and et al., "Coordination for multi-robot exploration and mapping," in *Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, 2000. 115

[76] Y. Yabu, M. Yokoo, and A. Iwasaki, "Multiagent planning with trembling-hand perfect equilibrium in multiagent POMDPs," in *Agent Computing and Multi-Agent Systems*, vol. 5044 of *Lecture Notes in Computer Science*, pp. 13–24, Springer Berlin / Heidelberg, 2009. 115

[77] F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *J. Artif. Int. Res.*, vol. 32, no. 1, pp. 289–353, 2008. 115

[78] S. Hauert, J.-C. Zufferey, and D. Floreano, "Evolved swarming without positioning information: an application in aerial communication relay," *Auton. Robots*, vol. 26, no. 1, pp. 21–32, 2009. 116

[79] P. Basu, J. Redi, and V. Shurbanov, "Coordinated flocking of UAVs for improved connectivity of mobile ground nodes," in *Military Communications Conference, 2004. MILCOM 2004. IEEE*, vol. 3, pp. 1628 – 1634 Vol. 3, 31 2004. 116

[80] Y. Z. Lee, *Scalable ad-hoc routing: design and implementation*. PhD thesis, 2008. Adv.-Gerla, Mario. 116

[81] E. W. Frew and T. X. Brown, "Networking issues for small unmanned aircraft systems," *Journal Intell. Robotics Syst.*, vol. 54, no. 1-3, pp. 21–37, 2009. 116, 126

[82] C. H. Liu, T. He, K.-w. Lee, K. K. Leung, and A. Swami, "Dynamic control of data ferries under partial observations," in *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*, pp. 1 –6, 18-21 2010. 117

[83] P. Scerri, T. Von Gonten, G. Fudge, S. Owens, and K. Sycara, "Transitioning multiagent technology to UAV applications," in *Proc. AAMAS'08*, pp. 89–96, 2008. 118, 135

[84] J. Ousingsawat and M. G. Earl, "Modified lawn-mower search pattern for areas comprised of weighted regions," in *American Control Conference*, pp. 918–923, 2007. 120

[85] D. Swihart, F. Barfield, E. Griffin, B. Brannstrom, R. Rosengren, and P. Doane 122

[86] P. DeLima and D. Pack, "Maximizing search coverage using future path projection for cooperative multiple UAVs with limited communication ranges," in *Optimization and Cooperative Control Strategies*, pp. 103–117, Springer, 2009. 135

[87] D. J. Pack, P. DeLima, G. J. Toussaint, and G. York, "Cooperative control of UAVs for localization of intermittently emitting mobile targets," *IEEE Trans. Systems, Man, and Cybernetics, Part B: Cybernetics.*, vol. 39, pp. 959–970, Aug. 2009. 135

[88] E. Model, "Pilatus pc-6 porter scale 260 (arf)," 2011. `http://www.espritmodel.com/pilatus-pc6-porter-scale-150-arf.aspx`. 148

[89] D. DRONES, "Ardupilot mega," 2011.
`http://code.google.com/p/ardupilot-mega/wiki/Xplane`. 150

[90] C. Partridge and R. Hinden, "Reliable data protocol." RFC 1151, 1990. 154

[91] A. Valera, W. K. G. Seah, S. Rao, and S. Rao, "Champ: A highly-resilient and energy-efficient routing protocol for mobile ad hoc networks," *IEEE MWCN*, pp. 79–85, 2002. 154

[92] D. DRONES, "Hardware in the loop simulation using the Ardupilot Mega and X-Plane," 2011. `http://code.google.com/p/ardupilot-mega/wiki/Xplane`. 157

[93] F. Ivis, "Calculating geographic distance: Concepts and methods," in *Proceedings of the 19th annual NorthEast SAS Users Group (NESUG) Conference*, sep 2006. 164