Linköping Studies in Science and Technology

Thesis No. 1476

Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles

by

Per-Magnus Olsson

Submitted to Linköping Institute of Technology at Linköping University in partial fulfilment of the requirements for degree of Licentiate of Engineering

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Linköping 2011

ISBN 978-91-7393-200-4, ISSN 0280–7971
Printed by LiU-Tryck, 2011

Electronic version available at :
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-66060

Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles

by

Per-Magnus Olsson

April 2011
ISBN 978-91-7393-200-4
Linköping Studies in Science and Technology
Thesis No. 1476
ISSN 0280–7971
LiU–Tek–Lic–2011:15

ABSTRACT

Surveillance is an important application for unmanned aerial vehicles (UAVs). The sensed information often has high priority and it must be made available to human operators as quickly as possible. Due to obstacles and limited communication range, it is not always possible to transmit the information directly to the base station. In this case, other UAVs can form a relay chain between the surveillance UAV and the base station. Determining suitable positions for such UAVs is a complex optimization problem in and of itself, and is made even more difficult by communication and surveillance constraints.

To solve different variations of finding positions for UAVs for surveillance of one target, two new algorithms have been developed. One of the algorithms is developed especially for finding a set of relay chains offering different trade-offs between the number of UAVs and the quality of the chain. The other algorithm is tailored towards finding the highest quality chain possible, given a limited number of available UAVs.

Finding the optimal positions for surveillance of several targets is more difficult. A study has been performed, in order to determine how the problems of interest can be solved. It turns out that very few of the existing algorithms can be used due to the characteristics of our specific problem. For this reason, an algorithm for quickly calculating positions for surveillance of multiple targets has been developed. This enables calculation of an initial chain that is immediately made available to the user, and the chain is then incrementally optimized according to the user's desire.

This work has been supported by CUGS (the Swedish National Graduate School in Computer Science), LinkLab (www.linklab.se), the Swedish National Aeronautics Research Program NFFP04–S4203 and NFFP05–01263, the ELLIIT Excellence Center at Linköping-Lund for Information Technology, the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII, the Center for Industrial Information Technology CENIIT (grant number 06.09) and the Linnaeus Center for Control, Autonomy, Decision-making in Complex Systems (CADICS), funded by the Swedish Research Council (VR).

Department of Computer and Information Science
Linköping universitet
SE-581 83 Linköping, Sweden

Acknowledgements

This thesis would not have been possible without the support of co-workers and friends. Especially I would like to thank:

My supervisors, professor Patrick Doherty and associate professor Jonas Kvarnström for guidance in research as well as in writing of this thesis.

My co-researchers Kaj Holmberg and Oleg Burdakov for many valuable discussions.

The colleagues at AIICS and especially Olov Andersson, Fredrik Heintz, David Landén, Martin Magnusson and Piotr Rudol for spending time reading this thesis as well as earlier papers.

Anne Moe for guiding me through the bureaucratic maze of graduate studies.

Other friends, too numerous to mention by name, for encouragement and inspiration.

Linda for being the world's best girlfriend.

Contents

Chapter 1

Introduction

Historically, Unmanned Aerial Vehicles (UAVs) have been used for tasks that are considered "dangerous, dirty and dull". Tasks can be dangerous if they require flying an aircraft over enemy positions. Dirty tasks may require entering areas contaminated by poison or flying into a radioactive cloud with the intention of collecting samples of radioactive dust. A dull activity is something that a human would quickly grow tired of doing.

An example of an activity that is often considered dull is surveillance, which is an essential aspect in a wide variety of applications, for example search and rescue operations, traffic monitoring, forest fire monitoring, law enforcement and military applications. Although mainly labeled as a dull activity, surveillance can also be dangerous, especially if the *surveillance target* is hostile, or if there is limited information about the area around the target. Improving the performance and decreasing the risk of human injuries and casualties are two of the many reasons for using UAVs for surveillance.

The use of unmanned vehicles for surveillance is not new. Such vehicles have been used throughout large parts of the twentieth century and the types of vehicles used vary greatly: from large semi-stationary airships, through UAVs a few meters in size, to micro UAVs weighing less than a kilogram. With advances in technology, the use of UAVs for surveillance as well as for other tasks is likely to increase.

In many cases, the information that is gathered by surveilling the target must be made available to a ground operator at a *base station* as quickly as possible. As the information may include high volume sensor data such as live video, high uninterrupted bandwidth is desirable. The communication equipment and the properties of the communication channel may restrict where the *surveillance UAV* can be placed. Naturally, it must be positioned in such a way that it can surveil the target, but it must also be able to

transmit the sensed information to the base station. To maintain good transmission quality for the high-bandwidth communications required when transmitting live video, common requirements are line-of-sight and limited distance between the surveillance UAV and the base station, corresponding to the maximum communication range [41].

The line-of-sight requirement can be problematic in mountainous or urban areas. While the problem can be mitigated by increasing the UAV's altitude, this option is not always possible for small UAVs, which in some cases are not able to ascend sufficiently. Although larger UAVs might be able to do this, the airspace may be restricted by aviation authorities, which in some cases makes it impossible for the UAV to achieve the required altitude.

If the UAV is able to ascend to the required altitude and that the airspace is available, the distance between the UAV and the target as well as between the UAV and the base station increases. This can adversely affect the quality of sensed information and the maximum communication distance may also be exceeded. Even if the transmission range is sufficient, communicating to and from such an altitude might require significant transmission power, which can be problematic for smaller UAVs. The communication range is also typically limited, and can be quite short, especially when smaller and lower cost UAVs are used. This is because such UAVs might not be able to carry the most powerful and sophisticated communication equipment due to size and weight constraints.

If UAVs are unable to ascend to sufficient altitude, another alternative is to use satellite communication. However, not all organizations have access to such satellites and smaller UAVs might not be able to carry the required equipment.

The above methods for achieving communication between the surveillance UAV and the base station are suitable in some situations, but there are limitations to both methods. An alternative approach is to use one or more communication relays that extend the effective range and forward transmissions around obstacles.

If it is known beforehand where the surveillance target will appear, then the necessary relays can be placed in advance. If the target location is unknown, then a large number of relays can be prepositioned to cover all possible target positions. However, this would probably require many statically placed relays, most of which would not be used, and it limits surveillance to environments where relays are expected to remain for some time.

A more flexible solution is to use UAVs to relay information. This has previously been investigated by several researchers, see e.g. Cerasoli [20] and Pinkney et al. [81]. However, there has been very little research on *where* the *relay UAVs* should be placed.

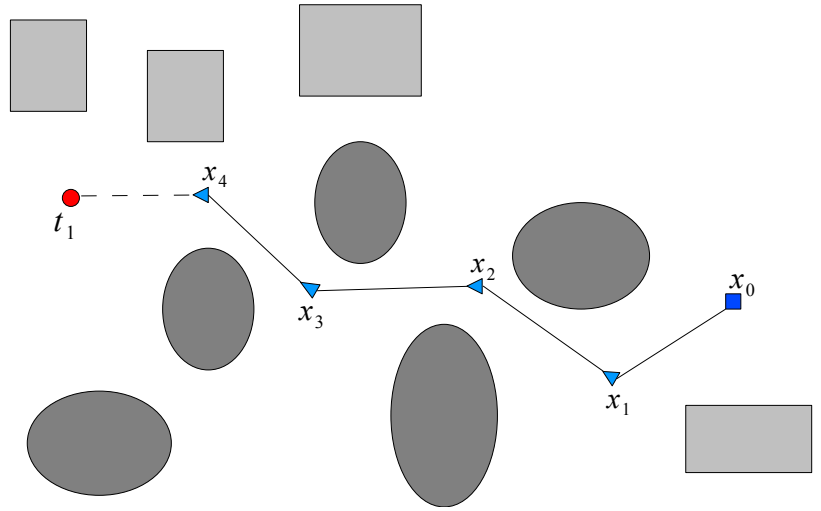


Figure 1.1: Example of relay chain with four UAVs. The base station located at location x_0 is connected to the target located at t_1 by the surveillance UAV at x_4 and the relay UAVs at x_1–x_3.

This thesis focuses on algorithms for finding suitable positions for such UAVs and gives examples of some of the factors that can be used to distinguish between good and bad positions with regards to UAV placement. For practical reasons, the surveillance UAV is distinguished from the relay UAVs. The surveillance UAV must be equipped with sensors suitable for surveilling the target while the relay UAVs may carry less sophisticated sensors as their task is to relay information. If the relay UAVs are placed correctly, they offer a way to handle both the limited communication range and the line-of-sight requirement. As the distance between the base station and the targets can be quite long, many relay UAVs may be required. For this reason, we aim to describe and develop algorithms that scale well enough to quickly solve problems involving a large number of UAVs in large areas of operation.

In cases where a single target is surveilled, the UAVs form a *relay chain* (Figure 1.1) between the target and the base station. When there are several targets, calculating separate chains to each would not make the best use of resources. Instead, relays could receive information from several UAVs. This creates a *relay tree*. Such a tree has the root in the base station, the UAVs are the interior nodes and the targets are the leaves. An example is shown in Figure 1.2.

Here we are interested in surveillance of one or more static targets with

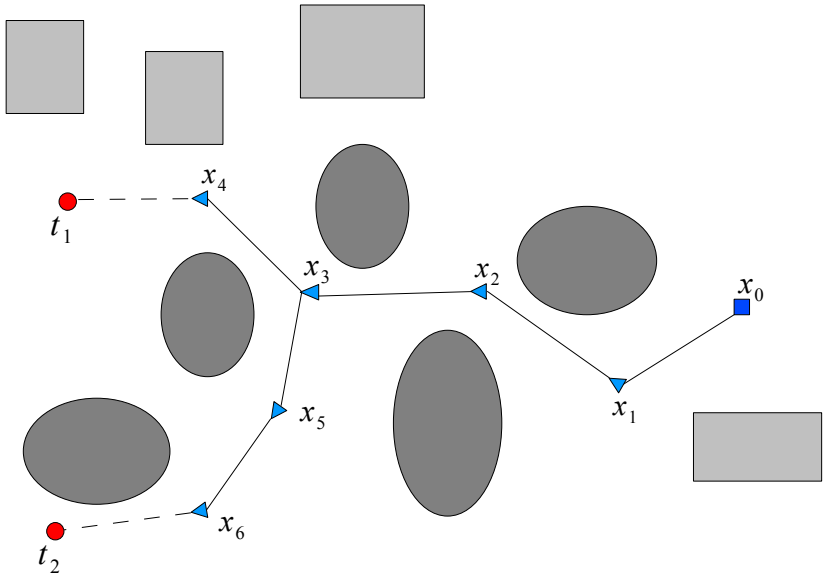


Figure 1.2: A relay tree with two targets.

known locations and as we are looking for positions where the UAVs can be placed, the relay problems are *positioning problems*, not motion planning problems. Algorithms for finding trajectories are not part of this thesis.

The focus in this thesis is on UAVs, especially helicopters that can remain at the same time for a prolonged time. However, the algorithms and concepts presented here work equally well for unmanned ground vehicles (UGVs) as well as for determining placement of other objects used for communication, for example temporary base stations for cellular phone communication in disaster situations.

1.1 Thesis Contributions

The main contributions of this thesis include:

- Formalizations of several single and multiple target relay positioning problems, focusing on different objectives and allowing for a large degree of flexibility in modeling surveillance and communication.
- Two different algorithms for solving the different single target relay positioning problems. Both algorithms are based on graph search in a discretized version of the original problem. Each algorithm calculates

a set of different relay chains, where each relay chain has a different quality and requires a different number of UAVs. Naturally, a solution requiring a larger number of UAVs is only useful if it has a higher quality. The first algorithm is a label-correcting algorithm that is capable of solving several different relay positioning problems. The second algorithm is focused on the problem of finding the highest quality solution given a limited number of available UAVs.

- A discussion on how the multiple target problems can be modeled to be solved efficiently as well as a theoretical study and discussion of what algorithms are suitable for solving the multiple target problems. We generalize an existing heuristic to fit the requirements of the multiple target relay positioning problems and present two other algorithms that can be used for different problems involving relay trees for multiple targets. The first of the algorithms for the single target relay problem is used to solve several different multiple target relay positioning problems involving a base station and two targets. The same algorithm is then generalized and used as a heuristic in order to improve existing relay trees with respect to factors such as the number of UAVs required to realize the tree or the quality of the tree.

1.2 Publications

Parts of this thesis have previously been presented in the following publications and reports:

[15] Oleg Burdakov, Patrick Doherty, Kaj Holmberg, Jonas Kvarnström, and **Per-Magnus Olsson**. Positioning unmanned aerial vehicles as communication relays for surveillance tasks. In *Proceedings of the 2009 Conference on Robotics: Science and Systems (RSS)*, pages 257–264, 2009.

[16] Oleg Burdakov, Patrick Doherty, Kaj Holmberg, Jonas Kvarnström, and **Per-Magnus Olsson**. Relay positioning for unmanned aerial vehicle surveillance. *International Journal of Robotics Research*, 29(8):1069–1087, 2010.

[17] Oleg Burdakov, Patrick Doherty, Kaj Holmberg, and **Per-Magnus Olsson**. Optimal placement of UV-based communications relay nodes. *Journal of Global Optimization*, 48(4):511–531, 2010.

[18] Oleg Burdakov, Kaj Holmberg, and **Per-Magnus Olsson**. A dual ascent method for the hop-constrained shortest path with application to positioning of unmanned air vehicles. Technical Report

LiTH-MAT-R-2008-07, Linköping University, Department of Mathematics, 2008.

[34] Patrick Doherty, Jonas Kvarnström, Fredrik Heintz, David Landén, and **Per-Magnus Olsson**. Research with collaborative unmanned aircraft systems. In *Proceedings of the Dagstuhl Workshop on Cognitive Robotics*, 2010.

[77] **Per-Magnus Olsson**, Jonas Kvarnström, Patrick Doherty, Oleg Burdakov, and Kaj Holmberg. Generating UAV communication networks for monitoring and surveillance. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2010.

The following publication has been submitted in 2010:

[19] Oleg Burdakov, Kaj Holmberg, and **Per-Magnus Olsson**. A dual ascent method for the hop-constrained shortest path with application to positioning of unmanned air vehicles. *Naval Research Logistics*, submitted in 2010.

1.3 Thesis Outline

Related work is discussed in Chapter 2. In Chapter 3, problem definitions for the single target relay problems are presented, as well as reachability functions for determining whether communication and surveillance can take place and cost functions for modeling the cost of such communication or surveillance. Different discretization options are discussed in Chapter 4. Several algorithms for solving relay positioning problems involving a single target are shown in Chapter 5. Chapter 6 defines the multiple target relay problems and presents algorithms suitable for solving the problems. An overview of our implementation and integration into a simulator system as well as experimental results are described in Chapter 7. The conclusions are presented in Chapter 8.

Chapter 2

Related Work

This chapter presents related work in the areas of relay placement and other related areas. Most previous work has been for UGVs, and the amount of work involving UAVs is somewhat limited.

As the problems of interest here are positioning problems, research in motion planning is not part of the related work. The general use of UAVs for surveillance is not included due to limited overlap with the research presented here.

The chapter is structured as follows. Section 2.1 provides an overview of the use of UAVs as relays. Next, Section 2.2 describes different ways to solve relay positioning problems for a single target. Different methods to provide relay trees to multiple targets are the topic of Section 2.3. Section 2.4 describes some different ways in which a UAV has been used to provide coverage of an area. Section 2.5 describes how ground robots have been used for exploration, while at the same time maintaining a connected network. Finally, Section 2.6 describes the relation between ad-hoc networks, wireless networks and the relay positioning problems. That section also provides an overview of how UAVs can be used in conjunction with ad-hoc networks.

2.1 UAVs As Relays

The concept of using a UAV as a communications relay in military applications is discussed in Pinkney et al. [81]. The work exclusively considers one UAV acting as a relay between users on the ground and the focus is on different classes of platforms and the different uses of those. Much emphasis is placed on the communications equipment, but no algorithm for UAV placement is presented.

A possible application of UAVs is highway surveillance, and the data link

used for transfer of information from a surveilling UAV to a base station has been investigated by Chen et al. [24]. Although it is mentioned that the UAV moves to keep large areas under surveillance, no algorithms for determining the placement of the UAV or the placement of the base station is performed: the focus is on the data link, including the hardware and its capabilities. An interesting point is that a commercial CDMA network is used for transmitting information from the UAV to the base station. Different bit rates and frame rates are tested to find the highest quality video stream that can be transmitted.

The research performed by Zhan [102] mainly focuses on the performance of communication between relays. However, there is a short discussion about placement of a UAV to enable transmission of information to several users. To communicate with a user, the UAV must be positioned in a circle centered in the user's position. The only described positioning algorithm is to place the UAV in the intersection of several circles. If no such intersection exists, then communication is not possible.

2.2 Single Target

In limited cases, a single relay UAV is sufficient to maintain a flow of information to the base station. One such case was investigated by Schouwenaars [86]. A surveillance UAV must fly to a specific position and a single relay UAV is used to maintain a connection with a base station. This problem was formulated as a Mixed-Integer Linear Programming (MILP) problem. The objective was to optimize a cost function while at the same time satisfying certain conditions. Several different conditions were used and presented. One condition was that UAVs were not allowed to fly within a certain distance from obstacles. Another condition was that the distance between the surveillance UAV and the relay UAV was required to be less than a maximum communication range. Similarly, the maximum allowed distance between the relay UAV and a base station was limited. The costs could for example be flight time, fuel consumption or visibility. Both centralized and distributed receding horizon approaches are considered. As the computational complexity increases exponentially with the number of agents, the distributed approach is used. For solving the MILP-problems, the commercial MILP-solver CPLEX was used. The same distributed approach was also used in a scenario with two relay UAVs [87]. It is mentioned that the time required in each iteration is increased when several relay UAVs are used, but the execution time was still within the allotted time interval.

Control behavior for teams of unmanned ground vehicles involving line-of-sight is investigated in Sweeney et al. [91]. In an indoor setting, a lead

UGV advances from the base station towards the goal position and incrementally determines where to place relay robots along the way in order to maintain communication with the base station. To enable communication between the robots, line-of-sight (LOS) between them is required and they must be within a certain distance from each other. LOS is estimated in a discretized environment with square grid cells of equal size. Two distances are used: *LOS distance* and *occlusion threshold distance*. If the distance between two robots is less than the LOS distance, then both robots can move freely, assuming that there is line-of-sight between them. The occlusion threshold distance marks the maximum allowed separation between a pair of robots and cannot be exceeded. If the distance between two robots is more than the LOS distance, the controller of one of the robots is switched off, and that robot remains passive until the other robot comes within LOS distance. By varying the occlusion threshold distance, and how proactive the robots are when trying to maintain line-of-sight, different behaviors are achieved.

A very similar problem is investigated by Nguyen et al. [73]. Several algorithms for positioning UGVs to form a relay chain between the base station and a target are evaluated in terms of energy usage. Initially, all robots are gathered at the base station and then the lead robot advances towards the target. The best-performing algorithm, with respect to energy usage, keeps the other robots at a base station until the lead robot experiences a signal strength below a threshold. When this happens, the lead robot stops and requests a relay UGV to be placed at the lead UGV's position. The relay moves from the base station to the lead UGV's position and stays there. This allows the lead UGV to move incrementally towards the target until another position with poor radio signal is encountered. Then a new relay UGV moves from its current position at the base station to the already placed UGV. When it arrives, the already placed relay UGV moves to the position of the lead robot, which is then free to continue. This process is repeated until the target position is reached by the lead UGV. The main disadvantages with the algorithms mentioned above are that they have no theoretical guarantees that the target position is reached, as no a priori calculation or evaluation of paths is performed.

Cheng et al. [25] investigated the use of several UAVs for relaying information between a producer and a consumer. They consider the information to be delay-tolerant, which opens up the possibility of transmitting the information from the producer to one UAV, which then flies to the consumer and transmits the information. Two UAVs are used simultaneously. One is delivering information from the producer and one is flying "empty" to the producer to get the next load of information. The authors refer to this

procedure as "load-carry-and-deliver". As we consider the information from the surveillance UAV urgent, this approach cannot be used for the applications discussed in this thesis. However, if the number of relay UAVs is insufficient to form a relay chain, this is one method to get some, albeit delayed, information from the surveillance UAV.

2.3 Multiple Targets

Finding suitable positions for UGVs to allow them to find relay chains between a set of sensors (targets) and a base station using potential fields is investigated by Simonetto et al. [89]. The UGVs follow the gradient of the total potential field until a suitable position is found. Several potential field approaches are evaluated, both "standard" and "dynamic". In the "standard" potential field approach, all robots are influenced by all other robots as well as the sensors. In a "dynamic" approach, the robots change the potential fields to influence other robots to position themselves in configurations that form chains between the sensors and the base station. Each UGV is affected by all other UGVs and sensors within a certain range. Several different environments are used to evaluate the approaches with respect to connectivity and efficiency. In all environments, the "dynamic" approach performs the best as the "standard" approach causes the robots to spread evenly in the environment. This is disadvantageous as it does not necessarily places robots at locations that are good for forming relay chains.

If a large set of targets must be visited and the number of available UGVs is smaller than the number of targets, then the UGVs must move between targets while at the same time maintaining communication with the base station. One possibility is to create a tree rooted in the base station and spanning all targets and visit one target at a time [72]. Several different tree types are possible, such as depth-limited trees or minimal spanning trees, or trees based on a traveling salesman tour. The trees are evaluated with respect to different criteria, for example average travel distance for each robot. Communication between robots is modeled using a virtual spring-damper model. If the robots are so far away from each other that the signal quality decreases below a threshold, the robots are attracted towards each other. This supposedly avoids the risk of disconnection, although few details are provided.

Another option that also builds on the concept of a tree spanning all targets is to divide the robots into groups, and let each group visit all targets in a subtree. Depending on the number of robots in a group and the number required to visit all targets, it might be possible to visit several targets simultaneously [71]. As the problem is shown to be NP-hard, dif-

ferent heuristics are evaluated with regards to the same criteria as for the sequential traversal method. The heuristics differ in when to split a robot group and the order in which targets are visited when sequential visits are necessary. In our problems, we assume that we have access to enough UAVs to surveil all targets concurrently, therefore the need for route planning between surveillance positions is removed.

2.4 Area Coverage

A general investigation of whether using a single relay UAV could improve communication in a simulated urban environment has been performed by Cerasoli [20]. Here the UAV works as a relay between users on the ground and line-of-sight is required between the users and the UAV for communication to take place. The focus of the work is to determine the percentage of the urban area that can be covered with acceptable signal strength, using a single UAV. Eight different positions on a circle, as well as in the middle of the circle are evaluated. The altitude is also varied, between 500, 1000 and 2000 meters. When the UAV is placed at a higher altitude, it has line-of-sight to a larger percentage of the area, and even though the signal strength decreases, better coverage of the area is achieved.

Han et al. [47] performed similar experiments, by investigating the effect of using a UAV to improve the global message connectivity and worst-case connectivity in a network. A series of experiments were performed to investigate the impact that the UAV had on the two types of connectivity. The size of the area was fixed and the number of users was varied between 2–30. As the number of users grew, connectivity improved as more users could communicate without the use of the UAV. Thus, the greatest improvement was achieved when the number of users was small. However, in all cases the UAV could significantly improve both connectivity measures.

The research in the above papers has little in common with the problems that we are interested in. We know the locations of the surveillance UAV and the base station and are consequently not interested in providing coverage of a large area. Neither are we interested in providing communication between arbitrary users.

2.5 Exploration

Exploration is one of many application areas for robots. When several robots cooperatively explore an area, it can be very beneficial if they are able to exchange information during the exploration process [83]. The problem

is discretized, both temporally and spatially, using a grid. A grid cell is considered explored when it has been visited by a robot. An algorithm based on maintaining an exploration frontier is used to weigh the benefits of exploring unknown territory versus maintaining communication with other robots. A set of possible moves is determined for each robot, and all moves are evaluated with a utility function. The function evaluates each move with respect to different factors, for example, whether it explores a cell on the exploration frontier and whether it maintains communication with other robots. An optimization is performed in each time step, to determine what actions yield the highest total utility for the team. This problem is different from our problems: we are interested in finding a set of locations where robots will be *placed*, rather than finding a sequence of positions that allow robots to explore an environment.

Anderson et al. [5] presents an algorithm for maintaining line-of-sight between groups of ground robots exploring an area. The algorithm is based on several heuristics. First, existing groups of robots are identified, and then groups are connected to each other using a set of relays with the estimated lowest cost to connect the groups. Robots acting as relays are placed one at a time, until a certain confidence threshold is reached. The confidence threshold indicates that any solution found is of sufficiently high quality. Simulated testing indicates that the algorithm performs well, but the algorithm has no theoretical completeness guarantee.

Arkin and Diaz [6] use a behavior-based architecture to allow teams of ground robots with line-of-sight communication to explore buildings and to find stationary objects using only limited knowledge about the area in which the objects are placed. The objective is somewhat different from the objective in this thesis, as the task is to find effective exploration strategies that minimize the time required until all objects are found.

2.6 Ad-hoc Networks and Wireless Sensor Networks

Problems that are seemingly very similar to the relay positioning problems are encountered in ad-hoc networks. In such networks, messages are to be delivered in a network where there is no control of the network topology. Routing algorithms for such networks must be able to handle addition and removal of nodes at runtime [55, 62, 67].

The use of a swarm consisting of several UAVs to improve the range and reliability of an ad-hoc network is investigated by Palat et al. [79]. Good results are achieved, mainly through a large increase in the range using the

same transmission power, compared to using a direct ground link.

Brown et al. [14] investigated different scenarios where a UAV was used as a relay node. In the first scenario, there were two groups of radio nodes on the ground. Each group could communicate internally, but could not communicate reliably with the other group. In this scenario, the UAV allows the groups to form a functioning ad-hoc network and both throughput and connectivity were significantly improved.

The second scenario tested whether a UAV could improve throughput between moving radio nodes, including other UAVs. The result was that the UAV greatly improved the connectivity for nodes having initially poor connectivity while the connectivity was somewhat adversely affected for nodes with good initial connectivity. The authors speculate that this depends on a more variable link quality when using UAVs. To determine the subjective quality, they tested web browsing and a real-time voice application via an ad-hoc network. For web browsing, the performance was acceptable when using ad-hoc networks with up to six hops. The voice application worked well with up to three hops.

Significant differences exist between the relay problems and ad-hoc networks. In the relay problems, we are not interested in communication between arbitrary nodes, but in transferring large amounts of information between the surveillance UAV(s) and the base station. For this reason, we are not interested in maintaining connectivity between arbitrary nodes. Instead, we are interested in where the nodes (UAVs) should be positioned so that information can be transmitted to the base station. Furthermore, we have control over the placement of the UAVs and assume that the UAVs will be available for the complete mission.

Wireless Sensor Networks (WSNs) consist of a large number of small sensors that are placed to cover an area [2]. For a survey of routing algorithms in WSNs, the reader is referred to Al-Karaki and Kamal [3]. Although there are some similarities with the problems investigated in this thesis, there are also considerable differences: WSNs must be able to handle frequent sensor failures, and relays are often also sensors and should be placed accordingly. In WSNs, there is also limited control over where the sensors are placed.

Chapter 3

The Relay Positioning Problems

In this chapter, we provide definitions of several different variations of relay positioning problems, and we discuss factors and functions that can be used to determine whether communication and surveillance are possible. We also discuss how the cost of such activities can be modeled. A discussion about continuous solution methods is also included in this chapter.

3.1 Problem Setup

We assume that relays are placed in three dimensions. Let $F \subseteq R^3$ be the region that is free from obstacles, defining the space through which free line-of-sight can be achieved. Let $U \subseteq F$ be the region where each individual UAV may safely be placed. This region must only include points sufficiently far away from obstacles for the required safety clearances to be satisfied. *No-fly-zones* where UAVs are not permitted may also be excluded from U. Let $x_0, t_1 \in R^3 \setminus U$ be the position of a base station and a surveillance target, respectively.

Assume as given two Boolean reachability functions: a *communication reachability function* $f_{comm}(x, x')$ and a *surveillance reachability function* $f_{surv}(x, x')$. The communication reachability function specifies whether communication between two entities at points $x, x' \in U$ should be considered feasible. It can for example be defined by a limited communication radius and a requirement of free line-of-sight (where all points between x and x' must be in F), by explicit models of 3D wave propagation, or by any other definition appropriate for the problem at hand. The surveillance reacha-

bility function $f_{surv}(x, x')$ specifies whether a surveillance UAV at $x \in U$ would be able to surveil a target at $x' \in R^3 \setminus U$. This function must take into account suitable minimum and maximum ranges for surveillance as well as sensor-specific limitations such as cameras that cannot surveil targets in arbitrary directions. For example, a camera mounted on the belly of the UAV cannot surveil targets above the UAV.

Under the assumption that the corresponding reachability function holds, a *communication cost function* $c_{comm}(x, x')$ determines the cost of communication from x to x' and a *surveillance cost function* $c_{surv}(x, x')$ determines the cost of surveilling a target at x' from the position x. The general notion of cost and cost minimization can be used to model a wide variety of quality measures or combinations of such measures. For example, communication costs may be related to transmission power requirements, the risk of interrupted communication or intermittent dropouts, and the risk of detection by adversaries.

Several relay problems for a single target will now be defined. Multiple targets are treated in Section 6.1.

3.2 Definitions of the Single Target Relay Problems

A *relay chain* π between the base station position x_0 and the single target position t_1 is defined as a sequence of positions $[x_0, x_1, \ldots, x_k, t_1]$, where $\{x_1, \ldots, x_k\} \subseteq U$, such that $f_{comm}(x_i, x_{i+1})$ for all $i \in [0 \ldots k-1]$, and $f_{surv}(x_k, t_1)$. The *length* of a chain is defined as the number of agents required to realize the chain, including the base station and all UAVs: $\mathit{length}([x_0, x_1, \ldots, x_k]) = k + 1$.

We are often interested in generating relay chains of high quality relative to a problem-specific quality measure. We model such measures in terms of the *cost* of relaying information between positions x_i and x_{i+1} and the cost of surveilling the target at point t_1 from a surveillance UAV at point x_k.

The *cost* of a relay chain $[x_0, x_1, \ldots, x_k]$, denoted by $\mathit{cost}([x_0, \ldots, x_k])$, is defined as

$$\sum_{i=0}^{k-1} c_{comm}(x_i, x_{i+1}) + c_{surv}(x_k, t_1)$$

Given a problem instance as defined above, including the position of the base station and the target, we can now identify a number of interesting single target relay positioning (STR) problems. Some of these problems assume an upper limit on the number of UAVs available, denoted by M. Setting $M = \infty$ requires finding all solutions, regardless of length.

STR-MinLengthMinCost: Find a relay chain of minimum length among the chains of minimum cost. A solution to this problem is a chain s such that for all other chains c, $cost(s) \leq cost(c)$ and $cost(s) = cost(c) \rightarrow length(s) \leq length(c)$. This corresponds to using the highest quality chain that can be realized with access to an unlimited number of UAVs, with a preference for using fewer UAVs if this is possible without compromising quality.

STR-MinCostMinLength: Find a relay chain of minimum cost among the chains of minimum length. A solution to this problem is a chain s such that for all other chains c, $length(s) \leq length(c)$ and $length(s) = length(c) \rightarrow cost(s) \leq cost(c)$. This is useful if minimizing the number of UAVs is strictly more important than maximizing quality.

STR-MinCostLimited: Find a relay chain of minimal cost among the chains that use at most M UAVs. A solution to this problem is a chain s such that for all other chains c, $length(s) \leq M + 1$, and for all other chains $length(c) \leq M + 1 \rightarrow cost(s) \leq cost(c)$. This corresponds to a desire to find the highest quality relay chain that can be realized within the given limit on the number of UAVs.

STR-ParetoLimited: Find a set of *Pareto-optimal* relay chains that is complete up to a given upper limit on the number of available UAVs. A chain s is Pareto-optimal for up to M UAVs if $length(s) \leq M+1$ and for all chains c of length at most $M + 1$, $length(c) < length(s) \rightarrow cost(c) > cost(s)$ and $cost(c) < cost(s) \rightarrow length(c) > length(s)$.

The **STR-ParetoLimited** problem is a bi-objective problem, where each chain represents a different trade-off between the number of UAVs in the chain and the cost of the chain. Each such chain is *Pareto-optimal*, as it cannot be improved in one aspect without a decrease in another aspect [69]. For example, the cost of the chain cannot be improved without also increasing the number of UAVs in the chain.

Algorithms for solving the single target relay problems are discussed in Chapter 5.

3.3 Cost Functions

The purpose of a cost function is to evaluate pairs of positions with respect to how suited they are to place UAVs for communication or surveillance. A pair of better suited positions has a lower cost.

For two positions $x, x' \in R^3$, the communication cost function $c_{comm}(x, x')$ models the cost of communicating from x to x' and the surveillance cost function $c_{surv}(x, x')$ models the cost of surveilling a target located at x'

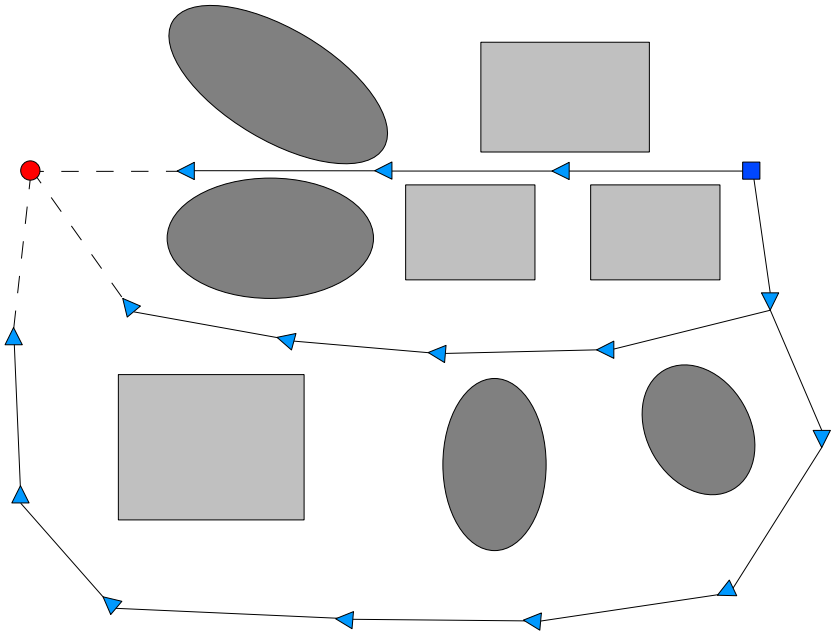


Figure 3.1: Three relay chains with different lengths and costs.

from position x. The functions are defined under the condition that f_{comm} and f_{surv} (see Section 3.4), respectively, hold for the positions x and x'. In this section, we first discuss several different measures for communication cost and then we discuss surveillance cost measures.

One straightforward measure of a relay chain's quality is the number of UAVs in the chain. However, in many cases it may be advantageous to use a larger number of UAVs if this can improve some other quality measure. An example is shown in Figure 3.1, where positions close to obstacles have higher cost as they offer a lower margin of safety and robustness to external factors. The UAVs in the top relay chain are positioned closer to obstacles and there is a small margin of safety around the UAVs' locations. On the other hand, the bottom relay chain uses a larger number of UAVs, but has a greater margin of safety between the UAVs and obstacles. The middle chain offers an intermediate between the two extremes.

Many different factors can be used as cost measures for positions. A cost measure using a combination of factors can be created using for example a weighted sum.

Some of the cost factors suggested in subsequent sections satisfy the *triangle inequality*, defined as $c_{x,x''} \leq c_{x,x'} + c_{x',x''}$ for all positions x, x', x'',

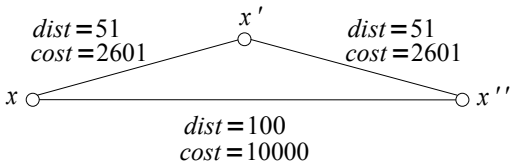


Figure 3.2: In some cases, e.g. if the cost increases quadratically with distance, then the triangle inequality no longer applies. This makes a direct transmission from x to x'' more expensive than two shorter transmissions.

where $c_{x,x'}$ denotes the cost of communication from x to x'. For such a function, communicating directly from x to x'' cannot be more expensive than first communicating to x' and then to x''. However, even simple cost functions can void the triangle inequality and as we will see, many cost functions of interest here do exactly this. Figure 3.2 shows three positions and the distances between them. Assume that the cost of transmission between positions increases quadratically with distance. An example of such a cost is the transmission power required to achieve a certain signal-to-noise ratio, as will be discussed in Section 3.3.1. With the cost set to the square of the distance, $c_{x,x'} = c_{x',x''} = 51^2 = 2601$ and $c_{x,x''} = 100^2 = 10000$. Thus $c_{x,x''} = 10000$ and $c_{x,x'} + c_{x',x''} = 5202$ which shows that relaying information through x' gives a cheaper path than transmitting directly to x''. The fact that the triangle inequality is not necessarily satisfied has profound implications for the relay positioning problems as it opens up the possibility of improving the quality of a chain by taking a longer path.

3.3.1 Transmission Quality

A surveillance UAV sending a continuous video feed to a base station may not be able to re-transmit lost or faulty data packets. Instead, a continuous stream of video data may be transmitted through the relay chain, where forward error correction [63] is used to recover from errors. However, it is likely that it is impossible to recover from some errors, which decreases the quality of the video stream received at the base station. Such a loss of quality is naturally modeled as a communication cost.

Obviously, the risk of such errors increases as the signal strength decreases. To model this, one can use a cost function inversely related to the *signal-to-noise ratio* (SNR). The SNR is one way of measuring signal strength. When calculating the SNR between two positions, the first step is often to determine the distance between the positions and whether there is *line-of-sight* between the positions. These two factors have a large impact

on the communication quality. The SNR is proportional to the transmitter power P and the antenna gain W and inversely related to the distance between transmitter and receiver $d = \|x' - x\|$. How much the signal-to-noise ratio decreases with distance is approximated by the *path loss exponent* α [79].

The path loss exponent is a very generic factor that is commonly used to approximate different factors. For example, stronger fading due to lack of line-of-sight yields a higher value of α. When used to estimate fading outdoors, the value of α is most often in the 2–5 range, where $\alpha = 2$ corresponds to propagation in free air, that is, with line-of-sight. As the exact value of α depends on for example altitude, the amount of particles in the air, atmospheric conditions and the buildings in the environment, $\alpha = 4$ is commonly used if the exact value is unknown. The reader is referred to Palat et al. [79] for a discussion on different values of α.

As a high SNR shall yield a low cost, one possible communication cost function based on the SNR is:

$$\textit{SNR: } c_{comm}(x, x') = \frac{\|x - x'\|^{\alpha}}{PW}$$

In situations where UAVs with different communication equipment are used, the values of P or W can be set to the lowest values of transmitter power and antenna gain, respectively, to provide a pessimistic estimate of the SNR. More information about calculating transmission signal strength in urban environments, albeit with a focus on cellular phones, is available in Wagen and Rizk [99]. If the environment is known in advance and time allows, a more accurate model of transmission quality can be derived where factors such as reflection and absorption can be taken into account in the cost function c_{comm} [90, 95].

Another possibility is to set the communication cost to a constant if the distance is below some predetermined value, signifying that the probability of unrecoverable errors caused by a poor signal-to-noise ratio is very low at such distances. At longer distances, the communication cost can for example be related to the inverse of the SNR.

3.3.2 Position Visibility

Suppose that a relay UAV is able to relay information from more than one UAV at a time. UAVs can then participate in multiple concurrent surveillance missions. Even if only one mission is initiated at a time, we might still prefer to place relays in locations where they are also likely to be useful in the event that additional missions are initiated in the near future. Such missions may then be able to use fewer additional UAVs by connecting

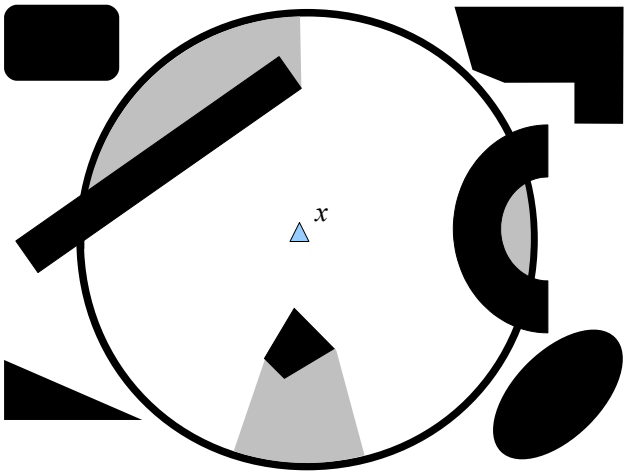


Figure 3.3: One possible cost measure is to use the *obstructed volume* in a sphere as cost. In this two-dimensional example, the cost would be the sum of the obstacles, drawn in black, and the non-visible volume, drawn in grey, inside the circle.

to an existing relay UAV rather than to the base station, which may be considerably further away.

A UAV can theoretically communicate with other UAVs in a sphere whose radius is equal to the maximum communication distance (see Section 3.4). Given line-of-sight requirements, parts of this volume may be obstructed by obstacles, as shown in Figure 3.3.

Let $V_{ob}(x)$ denote the obstructed volume from position x within the communication range r_{comm} and let $V_{comm} = \frac{4\pi {r_{comm}}^3}{3}$. Then the communication cost based on obstructed volume can be calculated as:

$$\textit{Obstructed volume: } c_{comm}(x, x') = \frac{V_{ob}(x)}{V_{comm}}$$

The position visibility cost measure is especially suitable for the multiple target relay problems, which are discussed in Chapter 6.

3.3.3 Minimum Free Angle Between Positions

The *minimum free angle between positions* gives a measure of the ability to perform transmissions between two positions even if wind or other factors

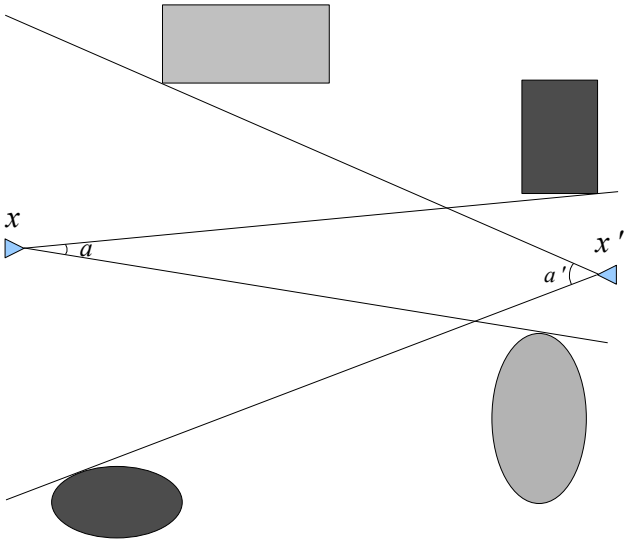


Figure 3.4: Minimum free angle between positions is a possible measure of cost.

affect the UAVs' ability to stay at the designated positions. The minimum angle places the greatest constraints on the UAVs' ability to transmit information and surveil the target the most. For this reason, it is a relevant cost measure when judging the suitability of positions for UAV placement.

In Figure 3.4, a UAV located at x must not move outside the cone originating in x' with angle a' and vice versa. The cost of communication between such positions should be inversely related to the size of the angle, i.e. a small angle should have a large cost. However, this might not be enough, as sufficiently small angles will cause great problems when trying to communicate as the UAVs become very sensitive to external factors such as wind. For example, two UAVs with a minimum free angle of 5° between them may be *more* than twice as sensitive to disturbances compared to if the minimum free angle was 10°. Therefore, to severely penalize very small angles, a measure such as the inverse of the logarithm of the minimum angle or some other non-linear function can be used as cost. In some cases, we may set the cost to zero or some small constant if the minimum angle is above some threshold value. For example, it might be extremely unlikely that the UAVs cannot stay inside a cone with a 30° angle, so any pair of positions with a minimum angle larger than 30° could have cost zero.

Thus, one possible communication cost function based on the minimum

angle between obstacles is:

$$\textit{Minimum angle: } c_{comm}(x, x') = \begin{cases} \frac{1}{min\{a,a'\}} & \textit{if } min\{a, a'\} < 30° \\ 0 & \textit{if } min\{a, a'\} \geq 30° \end{cases}$$

Which of these functions depending on the minimum angle is most suitable depends on the situation, where factors such as the environment, required safety margin and UAV susceptibility to external factors are taken into consideration.

3.3.4 Minimum Distance to Obstacles

Ideally, a UAV in U should be so far away from obstacles that there is no risk of collision regardless of external factors such as wind. In practice, this would very difficult to achieve, given that different vehicles are more or less sensitive to such factors and that some UAVs' control systems may not be capable of precise maneuvering. The set U would need to be calculated for each type of UAV, and even then, it would be very difficult to prove that no collision can occur. It could also lead to an unnecessary limitation of the search space. Due to the forbidden positions, many missions would require a large number of UAVs, and other missions would be impossible to perform. A reasonable compromise between safety margins and excessive limitation of available positions is to require that all positions in U have a minimum distance to obstacles, and generally prefer positions that are placed further away from obstacles. Such a requirement could be modeled as a cost, where for example the communication cost $c_{comm}(x, x')$ would depend on the distance from x' to the nearest obstacle.

With o as the closest obstacle from x', one possible cost function is:

$$\textit{Minimum obstacle distance: } c_{comm}(x, x') = \|o - x'\|$$

3.3.5 Surveillance Cost Functions

For the surveillance cost function, it can be relevant to use parameters from the sensors to estimate the quality of the sensed information and use such estimates in the surveillance cost function. If the sensor is a (video) camera, a position from which high-quality pictures can be taken would have a low surveillance cost. Such a position would have no intervening obstacles between it and the target and be located at an appropriate distance from the target.

It can also be beneficial to take factors other than the physical terrain into consideration when specifying the surveillance cost function. For ex-

ample, environmental and weather conditions, as well as the position of the sun, can be used. Low costs can be given to positions where the surveillance UAV's position will allow it to take pictures with a minimum of reflections and glare. If the visibility is poor due to for example fog, the surveillance cost could increase faster with increasing distance than if visibility is good. A similar cost function as for SNR (see Section 3.3.1) can be created by using the distance raised to some power α to signify that image quality decreases with distance, as well as with weather phenomena such as fog.

$$\textit{Distance to target: } c_{surv}(x, x') = \|x' - x\|^{\alpha} \quad \alpha \geq 1$$

The surveillance cost must be weighed against the communications cost for the rest of the chain. Unless the cost of surveillance is set high enough to influence the algorithms for finding solutions, a high-cost (low quality) surveillance position might be used as the surveillance cost only has a marginal effect on the cost of the complete chain.

3.4 Reachability Functions

The purpose of a reachability function is to determine whether communication or surveillance is possible between two positions. The reachability functions are Boolean functions that take as input two positions, x and x'. The communication reachability function $f_{comm}(x, x')$ holds only if communication between the two positions x, x' is possible. Similarly, the surveillance reachability function $f_{surv}(x, x')$ holds only if a UAV located at x can surveil a target located at x'. The factors used in the reachability functions are dependent on the application, and the algorithms to be presented in later chapters for solving the single and multiple target problems are independent of the function. As long as the functions have the above signatures, no further assumptions about the exact formulation of the functions are made.

We assume that the communication between several UAVs does not interfere with each other. This can for example be achieved through using techniques such as time-division multiplexing, frequency-division multiplexing and code-division multiplexing [95]. Time-division multiplexing means that transmitters take turns transmitting. Different frequencies for transmissions are used in frequency-division multiplexing and code-division multiplexing means that different encodings are used.

For the kind of transmissions of interest in this thesis, two common requirements are line-of-sight between sender and receiver and limited maximum transmission range. In particular, such requirements are very common

when transmitting high-volume sensor data, such as a live video feed, that requires high uninterrupted bandwidth. Formally, the line-of-sight requirement holds if $[x, x'] = \{x \in R^3, x' \in R^3 : \beta x + (1 - \beta)x', \beta \in [0, 1]\} \subseteq F$. That is, line-of-sight between two positions exists if a straight line between the two positions lies completely in the allowed set F. Note that we can allow $F \neq U$ as communication can go through areas where we might not be allowed to place UAVs. The limited communication range is formalized as $||x' - x|| \leq r_{comm}$. It is sometimes desirable to separate the UAV communication range from the UAV *surveillance range* r_{surv}. In that case, $f_{comm}(x, x')$ only holds if $||x' - x|| \leq r_{comm}$, in addition to any other requirements. Similarly, $f_{surv}(x, x')$ only holds if $||x' - x|| \leq r_{surv}$ in addition to any other requirements that are used.

Although line-of-sight is often required for high bandwidth communication, there are communication systems that do not require this. The Multiple Input Multiple Output (MIMO) communication system is an example of this [75, 44]. In fact, having line-of-sight could decrease the performance as objects in the environment are used to "bounce" the radio signals off. MIMO communication systems use several antennas for transmitting and receiving information, and the best performance is achieved if several independent signal paths are used. Such communication systems can also be modeled using the communication reachability function, although the line-of-sight requirement would most likely not be used.

Whereas f_{comm} depends on the properties of the communication system, the surveillance reachability function f_{surv} depends on the sensors used for surveillance. For cameras, common restrictions would be line-of-sight and a maximum range requirement, where the information sensed at the maximum range is of sufficient quality to be used, for example, a certain number of pixels per meter of target size for a camera. If several sensors are used to surveil the target, special considerations must be given to whether f_{surv} holds when one sensor senses the target, or when a certain number of sensors sense it.

The above are just examples of the many different ways to determine whether communication and surveillance is possible or not. No assumptions about the reachability functions are made anywhere else in the solution process and arbitrarily complex functions can be used to determine whether communication and surveillance can be performed, if desired.

3.5 Problem Properties

As the target and the base station can be placed arbitrarily and UAVs can be placed anywhere in the set $U \subseteq R^3$, the relay problems can be seen as

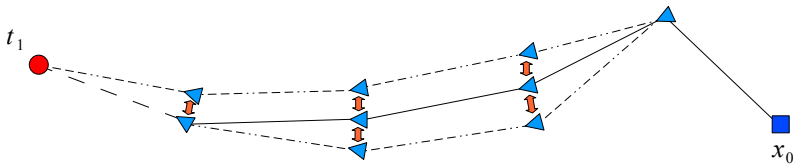


Figure 3.5: In an environment without obstacles or any limitations in the reachability functions, it is possible to transform any valid relay chain to any other valid relay chain, as indicated by the arrows.

continuous optimization problems. Here we discuss why current methods for continuous optimization are not practical for finding globally optimal solutions for the relay positioning problems.

Assume a given instance of a relay problem in an environment without any obstacles, with the position of a base station x_0 and the target t_1, as well as a number of UAVs. The reachability functions are based on free line-of-sight. From this, it is possible to calculate a set of feasible relay chains. The number of relay chains in this set is generally uncountable as UAVs may be positioned anywhere in a continuous space. It is possible to transform any feasible relay chain to any other feasible chain by moving the UAVs continuously, as exemplified by Figure 3.5.

However, we are more interested in environments with obstacles and possibly also restrictions in the reachability functions, e.g. limited communication range. Then discontinuities will occur as it is no longer possible to communicate between arbitrary positions and surveil the target from any position. In such cases, the feasible set consists of several disjoint subsets. Within each such subset, the UAVs in a relay chain may still be moved to form another relay chain. However, it is not possible to transform a chain in one feasible subset to another chain in another feasible subset without going through infeasible points. An example of this can be seen in Figure 3.6, where going from the relay chain on the top to a relay chain on the bottom requires going through the infeasible region in between.

The number of different feasible subsets may grow exponentially with the number of obstacles, which can lead to a large number of subsets in obstacle-rich environments. The large number of subsets leads to many local maxima, as each subset has at least one such point.

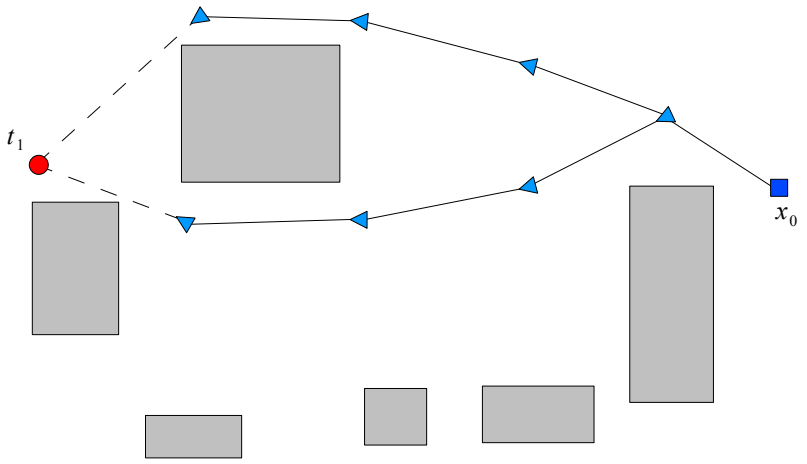


Figure 3.6: The continuous relay problems are computationally intractable as the feasible set is disjoint. A change from the top chain to the bottom chain, or vice versa, requires going through infeasible points.

3.6 Continuous Solution Methods

It is possible to use methods for continuous optimization to calculate relay chains. However, such methods are impractical to use due to the long execution time. In addition, it is an open research question how to formulate a requirement such as line-of-sight in terms of equalities and inequalities. Such a formulation is required in order to apply optimization methods for continuous (non) linear[1] problems. In general, it is an open research question how to solve problems such as the relay problems using methods for continuous (non) linear optimization, especially if we consider that the algorithms are to be used in a setting where a ground operator expects a quick response.

For each feasible set in which an initial chain has been found, continuous local optimization methods such as line search [74] can be used to find local extrema within that feasible set. Such methods calculate a step length and a direction in which a function value, such as the value of our cost function, improves. A new position is determined by moving a distance equal to the step length from the current position in the calculated direction. At the new position, it is checked whether the reachability function holds. If so, the new position becomes a new starting point for further search and the procedure can be repeated until the UAV is placed at a position that optimizes the function value of the cost function, subject to the constraints

[1] An optimization problem is linear if both the objective function and all constraints are linear, otherwise it is nonlinear.

in the reachability function. The process is then repeated for each UAV. This can create a locally optimal chain, but if we want to find a globally optimal chain, such a method must be executed for each feasible subset. Due to the large number of subsets, this would take prohibitively long time. Furthermore, for problems requiring finding a set of relay chains, such as **STR-ParetoLimited**, this requires repeating the process for all values of the number of available UAVs to up M.

Heuristics such as tabu search or simulated annealing lack the guarantee of finding the globally optimal chain. However, they can be used to find an initial relay chain in a feasible set and then continually improve it until a local extremum is found [45].

Another option for solving the relay problems is to use algorithms from the area of computational geometry, where continuous versions of Dijkstra's shortest path algorithm have been developed [70]. Such methods can be used for solving for example the **STR-MinLengthMinCost** problem. A disadvantage of these methods is that they approximate the environment using a set of surfaces and that the time complexity of the algorithm is dependent on the number of surfaces. Due to high time complexity, the execution time can be very long. In addition, problems such as **STR-MinCostLimited** involve finding several chains of different lengths and costs. The time required to solve such a problem suggests that other representations and methods are more appropriate.

3.7 Summary

In this chapter, several relay problems for a single target have been defined. Also, the use of reachability and cost functions have been explained, and several examples of cost functions have been given. The reachability functions determine whether communication and surveillance between a pair a positions is feasible. Therefore, they are often based on factors such as free line-of-sight and a limited maximum distance between the positions.

The purpose of a cost function is to evaluate the relative suitability of pairs of positions with respect to placing UAVs there for communication or surveillance and cost functions are often based on factors such as the distance to obstacles or the distance between the positions. Several of the cost functions void the triangle inequality, which makes it possible to decrease the cost of a relay chain by using a greater number of UAVs.

Obstacles in the environment as well as conditions such as limited communication and surveillance ranges and line-of-sight-requirements in the reachability functions divides the feasible set into several disjoint feasible subsets. Each such feasible subset has at least one local optimum. Deter-

mining the global optima using methods for continuous optimization can be very time consuming as such methods need to be executed once for each such subset.

Chapter 4

Environment Representation and Discretization

One way to decrease the execution time for finding solutions to the relay positioning problems is to use graph search algorithms to solve discrete approximations of the continuous problems. Such algorithms require a graph with nodes corresponding to positions where UAVs can be placed, and edges corresponding to potential communication or surveillance between positions. In this we chapter show how a discretization can be performed in order to construct a graph, and discuss different discretization strategies.

4.1 Discretization and Graph Creation

A discretization is performed with the intention of creating a graph consisting of a set of nodes, denoted by N, and a set of edges, denoted by E. $|N|$ denotes the cardinality for the set of nodes, and the notation is used analogously for other sets. A directed graph can be created as follows.

1. Determine a finite set of positions $U' \subseteq U$ among the free coordinates. These are the positions that will be considered valid placements for relay and surveillance UAVs in the discrete relay problems. Different ways to determine the set of positions U' are discussed in Sections 4.2–4.5.

2. Associate each position $x \in U'$ with a unique node.

3. Associate the base station position x_0 with a new node n_0.
4. Associate the target position t_1 with a new node τ_1.
5. For each $x \in U'$ corresponding to $n \in N$ and satisfying the communication reachability function $f_{comm}(x_0, x)$, create an edge $e = (n_0, n)$ of cost $c_{comm}(x_0, x)$ representing the possibility of communication between the base station and position x.
6. For each $x, x' \in U'$ corresponding to $n, n' \in N$ and satisfying the communication reachability function $f_{comm}(x, x')$, create an edge $e = (n, n')$ of cost $c_{comm}(x, x')$, representing the possibility of communication between positions x and x'.
7. For each $x \in U'$ corresponding to $n \in N$ and satisfying the surveillance reachability function $f_{surv}(x, t_1)$, create an edge $e = (n, \tau_1)$ of cost $c_{surv}(x, t_1)$ representing the fact that a surveillance UAV positioned at x would be able to surveil a target at t_1.

Then, the graph $G(N, E)$ represents a discretization of the continuous space. Note that the graph construction algorithm makes no assumptions about the reachability functions f_{comm} and f_{surv} other than their signatures.

In the discretized space, the *length* of a chain corresponds to the number of edges in the chain. This is also referred to as the *hop count*, where each edge corresponds to one hop. The *cost* of a chain is then defined as the sum of the edge costs along the chain. Thus, a *shorter* chain has fewer edges, while a *cheaper* chain has lower cost.

The discretization itself only allows the application of discrete optimization methods. The problem of finding a set of feasible UAV positions is still not solved. Instead, a reduction to a feasible problem is required. This can be done by considering the following: given a node n, the set of valid nodes for placing the next UAV in a chain can be identified as exactly the set of nodes for which $f_{comm}(n, n')$ holds. Let $N_n \subseteq N$ consist of all such nodes n', and analogously, $N_{n'}$ consists of all nodes all nodes for which $f_{comm}(n', n'')$ holds. Then, a complete chain from n_0 to τ_1 can be determined by starting from n_0 and choosing a node in N_{n_0} as the next node in the chain, and continuing recursively until a node n'' is found for which $f_{surv}(n'', \tau_1)$ holds. Recall that a UAV placed at such a position can surveil the target located at node τ_1.

From this, we know in theory how to choose the set of UAV positions so that a relay chain is formed. Naturally, we are interested in finding a high-quality (i.e. low cost) chain. Finding solutions to such problems is commonly done using graph search algorithms, which will be discussed further in Chapter 5.

In the remainder of this chapter, we discuss different options for choosing the finite set of discrete positions U' where UAVs may be placed. One of these methods places nodes with the same node density throughout the environment, while others distribute nodes unevenly, typically placing more nodes closer to obstacles. Which discretization method is most suitable depends on several factors, such as the number of nodes and their positions, the reachability functions, and the number of obstacles as well as how the obstacles are distributed in the environment.

Different graphs will affect the length and cost of the solutions. In some cases, a problem instance may require a large number of UAVs as the chain must go around areas with no or few nodes. There are several desirable characteristics in discretization methods that minimize the risk for this. To maximize flexibility in choosing the location of the next UAV in the chain, the nodes should have a high degree. The connected nodes should be placed at different angles and distances, providing good coverage of the area using few nodes. If this is not fulfilled, it might not be possible to communicate or surveil through gaps between obstacles, which can lead to long detours. To provide many different positions for the surveillance UAV and provide sensor information of high quality, it is also desirable to have a large set of nodes directly connected to the target node.

4.2 Fixed-Size Grids

A simple solution to choosing the set of potential relay and surveillance positions is to use a three dimensional grid with grid cells of equal size. The grid is placed over the terrain and a node is placed in each unobstructed grid cell. The node can be positioned anywhere in the cell, but for illustration purposes they are placed in the middle of each cell. Whether a grid cell is unobstructed can be determined in many different ways. For example, if there are no obstacles within a certain distance from the node position, then the cell could be considered unobstructed.

The resolution is a very important factor when using grids. If it is too low, then UAVs can only be placed at a few places, possibly causing unnecessarily long relay chains. Figure 4.1 shows a graph with nodes created from a medium-resolution grid, which is in contrast to the low-resolution grid displayed in Figure 4.2. In the coarser discretization, 6 UAVs are required, compared to 4 UAVs in the finer discretization as the maximum communication and surveillance range of each UAV can be better utilized.

Some types of terrains might require grids of high resolution to ensure that all positions can be reached, which leads to high memory consumption.

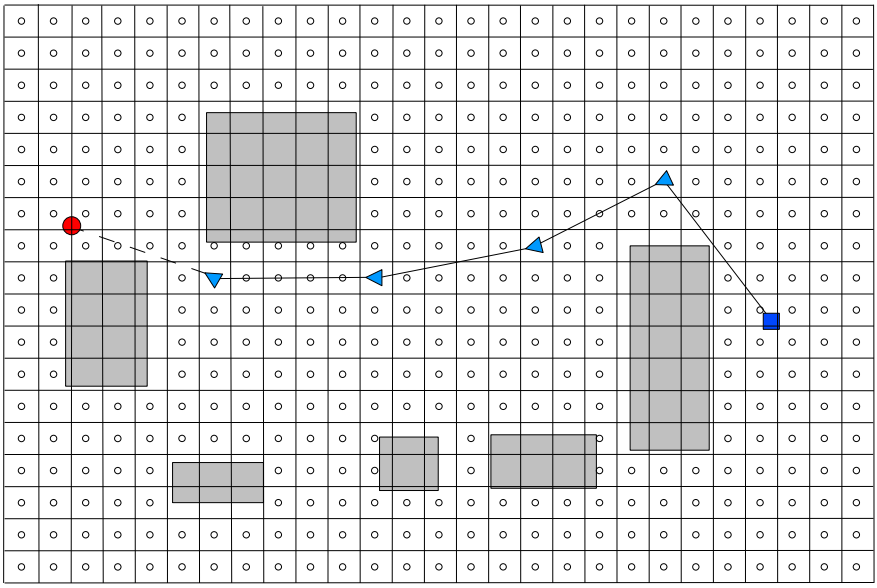


Figure 4.1: Nodes created using a medium-resolution grid. The size of the nodes have been exaggerated for illustrative purposes, and the edges are left out to reduce clutter.

4.3 Octrees

Assume that fixed-size grids are used in an environment that mainly consists of a large open area but also has some closely spaced obstacles. To assure that UAVs can be placed between the obstacles, the grid cell size must be quite small. The large number of nodes leads to large memory requirements, and the high resolution is not necessary in the large open area. To decrease the number of nodes and thus the memory requirements, an alternative is to use a data structure with cells of varying size. Examples of such structures are *quadtrees* (for two-dimensional environments) and *octrees* (for three-dimensional environments) [43, 49]. Many other data structures are possible, and an overview of such data structures for different uses is available in Samet [85]. Although data structures such as octrees are often used to subdivide areas spatially, for example to speed up geometry intersection checks, they can also be used to place nodes. A two-dimensional example is shown in Figure 4.3.

The octree is a hierarchical data structure, with recursive decomposition from the top level with the largest volumes, to the bottom level, which contains the smallest volumes. Volumes that are not further divided are called leaves. Each non-leaf volume in an octree contains eight smaller

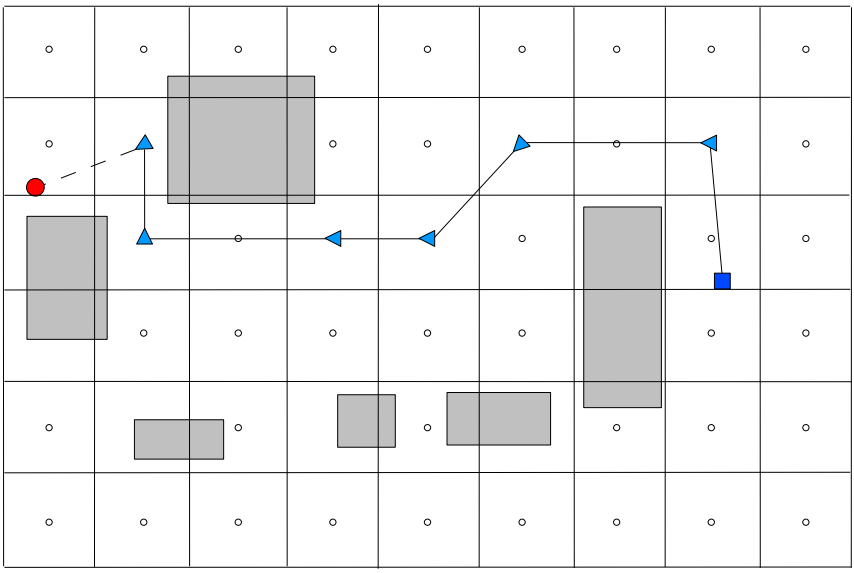


Figure 4.2: The low-resolution grid provides few possibilities for UAV placement, which leads to a longer relay chain.

volumes of equal size, organized in a $2 \times 2 \times 2$ fashion. Together the smaller volumes cover the exact same volume as the larger volume. Each of the smaller volumes, in turn, either is a leaf or contains eight even smaller volumes, and so on, until the desired number of levels, or the minimum cell size, is reached. Nodes can be placed for example in the middle of each volume or at the corners of the volumes.

Octrees are designed to cover cubic areas, and if the volume to be covered is not cube shaped, there are two possibilities. Either an octree covering a larger volume must be used, or the area must be divided into several cubic volumes.

Using a data structure with grid cells of varying size has the advantage of a decreased number of nodes and edges and therefore lower memory consumption. A disadvantage is the comparatively low concentration of nodes at positions far from obstacles, yielding fewer possible UAV locations.

4.4 Expanded Geometry Graphs

In an *expanded geometry graph*, obstacles in the environment are expanded a certain distance "outward", hence the name [64]. The exact distance varies, but is often related to the size of the UAVs and the required safety distance.

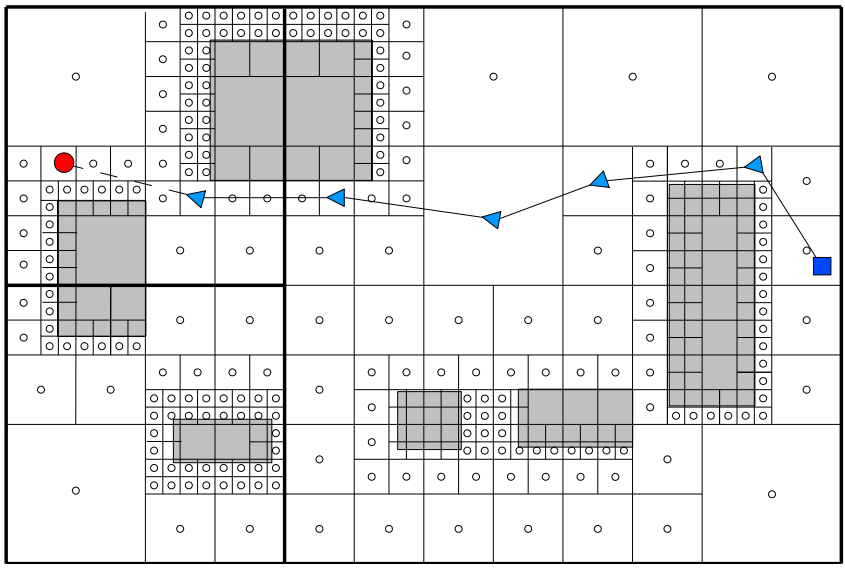


Figure 4.3: Several quadtrees covering an area. The heavy lines mark the edges of the quadtrees.

Nodes are then placed at the corners of the expanded objects. Figure 4.4 shows an expanded geometry graph. If obstacles are large, the distance between the nodes at its corners may exceed the maximum communication or surveillance range. In such cases, additional nodes might have to be placed along the sides of the obstacles to provide connections. The same problem can occur also occur in other cases, for example when obstacles are further from each other than the communication range or when the distance between the target position t_1 and the closest obstacle is longer than the surveillance range.

As nodes are placed close to the corners of obstacles, this approach is suitable for urban terrains, where the node placement also ensures that relay chains can use any gaps between buildings. However, this approach to discretization is somewhat brittle: if the terrain is so dense that the expanded obstacles collide, then no nodes can be placed. In addition, the approach is not suitable in terrain involving large open areas as no nodes are placed at such locations. The lack of nodes forces the relay chain to take long detours around such areas. Figure 4.4 shows an example of this.

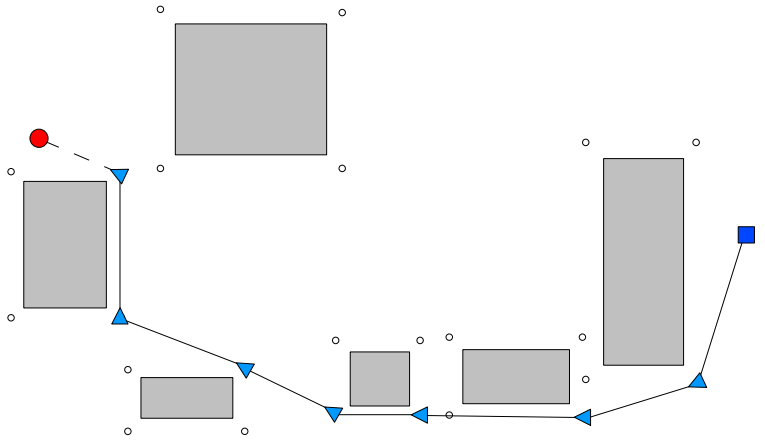


Figure 4.4: An expanded geometry graph with nodes placed at corners of objects.

4.5 Voronoi Diagrams

Another possibility is to use a graph created by a *Voronoi diagram* [7, 76]. The Voronoi diagram is a discretization that takes as input a set of obstacles. Then, the environment is divided into convex cells such that each cell contains exactly one obstacle and every point in a cell is closer to the obstacle in it than to any other obstacle. Figure 4.5 displays a Voronoi diagram, where the boundaries between the cells are drawn as lines. The boundaries are the positions that offer maximum distance between the obstacles. Nodes can then be placed where three or more cell boundaries meet [13]. This causes the number of nodes to be a function of the number of obstacles. Where there are few obstacles, there will be few nodes. If there were no obstacles, only a single cell would be created. This would generate very few nodes, in many cases making it impossible to find solutions to the relay problems. Therefore, this approach is best suited for environments with a large number of obstacles. To remedy the problem of too few nodes, one option is to also place nodes on the cell boundaries [11].

Voronoi diagrams have been used in many areas, for example UAV path planning [21, 9, 13, 68] and UGV path planning in environments with many buildings [11].

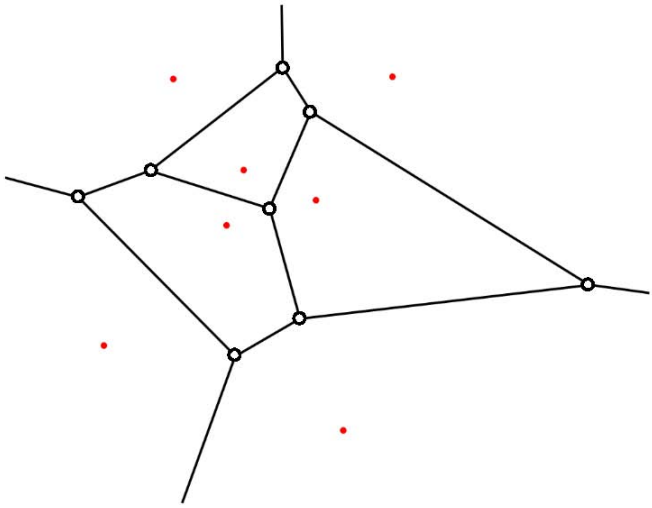


Figure 4.5: An Voronoi diagram with points as obstacles for illustrative purposes. A cell is created for each obstacle. The lines mark the boundaries between the cells. Nodes are placed where three cells meet.

4.6 Discretization Methods Used in Motion Planning

The research area of motion planning is concerned with finding paths from a start position to a goal position. Given a potentially high-dimensional space, discretizations are commonly used to make the problems tractable. Furthermore, algorithms commonly use discretizations created specifically for motion planning, rather than the general methods discussed so far. Examples of such methods are Rapidly Exploring Random Trees (RRTs) [59, 60, 80] and Probabilistic Road Maps (PRMs) [57, 60]. RRT algorithms construct a new tree every time motion planning is performed, while PRMs produce a graph in advance and add start and goal nodes each time a new motion planning request is performed.

Different methods for selecting the positions in RRTs and PRMs have been tried. Both simple methods such as sampling randomly within a distance and more advanced methods such as Gaussian sampling [12], bridge sampling [53], and sampling on the borders of obstacles [4] have been used.

In basic RRTs, each node is only connected to the predecessor in the tree and one or more successors. The main use of RRTs is to provide graphs for motion planning, and in many cases, a relatively small number of nodes of comparatively low degree is sufficient to provide coverage of an area.

In PRMs, a graph rather than a tree is created by connecting each node to all other nodes between which the correct reachability function holds, instead of just a subset of these nodes. Just like with RRTs, a relatively small number of nodes is often sufficient to cover an area.

A problem with using RRTs and PRMs to provide discretizations for the relay positioning problems is that they place nodes without any consideration to the number of hops that are required to reach the goal, as this is not a big concern in motion planning. Instead, the main concern is providing reasonably short paths with regards to distance, without any concern about how many nodes are used in the paths. This is very different from the relay positioning problems, where the number of hops in a path is very important, as the number of UAVs required to realize the path is proportional to the number of nodes in the path.

4.7 Summary

The continuous relay problems are computationally intractable as they typically have a disjoint feasible set and a large number of extreme points. In order to make the problems solvable in acceptable time, a discretization can be performed and the discrete problem is solved instead. A desirable characteristic of the graph is a large number of nodes at different distances and angles from each other. It is also advantageous that the nodes have high degree as this offers more options for communication and surveillance.

In this chapter, several different discretization methods have been discussed. The simple grid places nodes uniformly in the environment, while other methods such as octrees place more nodes closer to obstacles. Methods from motion planning, such as rapidly exploring random trees and probabilistic road maps can also be used, where the latter is more suitable due to higher node degree. In dense urban environments, the expanded geometry graph can be useful.

While all desirable characteristics may be fulfilled in some cases by a single discretization method, it is likely that a combination of methods will yield better results. As an example, an expanded geometry graph or a Voronoi diagram can be combined with a grid to provide a graph with a higher density of nodes at positions close to obstacles while at the same time maintaining a minimum node density further away from obstacles.

Chapter 5

Relay Positioning Algorithms for Single Target Problems

Once a discrete graph has been created, for example using any of the methods discussed in Chapter 4, graph search algorithms are used for solving the single target relay problems. This chapter discusses different algorithms for calculating relay chains, and experimental results for testing the algorithms with a fixed-size, three-dimensional grid are available in Chapter 7.

Both the **STR-MinLengthMinCost** and **STR-MinCostMinLength** (see Section 3.1) problems can be solved by a common shortest path algorithm such as Dijkstra's [32]. Alternatively, the A* algorithm can be used if a suitable heuristic is available [48]. Both problems require that optimization is first performed with respect to one factor and when the optimum of that factor is found, optimization is performed with respect to another factor. Though this may sound like a two-step process, it is commonly done using lexicographic ordering. For example, the **STR-MinLengthMinCost** problem can be solved for each node in N by using compound costs of the form $\langle cost(\pi), len(\pi)\rangle$ with $\langle cost(\pi_1), len(\pi_1)\rangle < \langle cost(\pi_2), len(\pi_2)\rangle$ if $cost(\pi_1) < cost(\pi_2)$ or ($cost(\pi_1) = cost(\pi_2)$ and $len(\pi_1) < len(\pi_2)$). Thus, chains are optimized first in order of cost and then in order of number of hops (UAVs) required. The **STR-MinCostMinLength** problem is solved analogously.

Since finding a relay chain from n_0 to τ_1 in a graph is equivalent to finding a path from n_0 to τ_1 in the same graph, we will use the terms interchangeably. As efficient algorithms for the **STR-MinLengthMinCost**

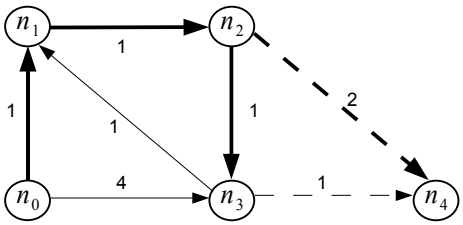


Figure 5.1: An example graph labeled with edge costs. The corresponding MLMC-tree is presented by heavy lines.

problem exists, the remainder of this chapter will explore algorithms for the **STR-MinCostLimited** and **STR-ParetoLimited** problems, beginning with the latter.

5.1 Existing Algorithms for the STR-Pareto-Limited Problem

STR-ParetoLimited is closely related to the *all hops optimal path problem* (AHOP) graph search problem, which has previously been applied in network routing problems [46]. For each $k = 1, 2, \ldots, L$, find a cheapest path from n_0 to τ_1 among those paths whose lengths do not exceed k. We call such paths AHOP solutions for the length k. From the definition of Pareto-optimality given in Section 3.2, it follows that any Pareto-optimal solution of length k is also an AHOP solution for length k. However, the reverse is not true: a Pareto-optimal solution of length k must also have minimal length among all paths of minimal cost. This is illustrated by the example in Figure 5.1. For $\tau_1 = n_4$, there is only one path of length 2, namely $\pi_1 = n_0 \to n_3 \to n_4$ with $cost(\pi_1) = 5$. This path is an AHOP solution for $k = 2$, since there is no cheaper path of length at most 2. Since there is also no equally cheap path of length strictly less than 2, it is also Pareto-optimal. There is one path of length 3, $\pi_2 = n_0 \to n_1 \to n_2 \to n_4$ with $cost(\pi_2) = 4$. Again, this is both an AHOP solution for $k = 3$ and a Pareto-optimal solution. However, consider the path $\pi_3 = n_0 \to n_1 \to n_2 \to n_3 \to n_4$, also of cost 4. Both π_2 and π_3 are AHOP solutions for $k = 4$, since they are of equal and minimal cost and are sufficiently short, but only π_2 is Pareto-optimal, since it is shorter.

Though the AHOP-problem and the **STR-ParetoLimited** problem are not identical, there is a close correspondence. Suppose that we have generated a complete set S of Pareto-optimal paths and require an AHOP

solution of length of at most k. If such solutions exist, the path in S having the greatest length $k' \leq k$ will be one of them. Conversely, suppose that we have generated a complete set of AHOP solutions. Then, a complete set of Pareto-optimal solutions can be found by grouping the AHOP paths by cost and selecting a path of minimum length in each group. Thus, the discretized **STR-ParetoLimited** problem can be solved by using an AHOP algorithm and selecting the Pareto-optimal solutions from the AHOP solutions.

As a solution to the **STR-ParetoLimited** problem commonly contains several chains, common shortest path algorithms such as Dijkstra's algorithm or A* cannot solve the problem by themselves in a single execution. An early method for calculating the set of solutions to the AHOP problem is the Bellman-Ford algorithm [29]. Given a start node, the algorithm finds paths requiring at most k edges during iteration k. If a cheaper path is found later, the more expensive path is discarded. This was later modified by Lawler [61], whose algorithm in iteration k stores the cheapest path for each node, using at most k edges, without overwriting any paths found in earlier iterations. Lawler's algorithm was further improved by Balakrishnan and Altinkemer [8], who introduced an upper bound on the number of edges so that only paths with length $k \leq L$ are calculated. This modification allows the algorithm to solve the AHOP problem more efficiently. The pseudocode of Balakrishnan and Altinkemer's algorithm is shown in Figure 5.2 and from now on, we will refer to it as Algorithm 1.

The following terminology is used for all algorithms. For all nodes n in N, $g_k(n)$ is the cost to reach node n from n_0 in at most k steps and $p_k(n)$ is the path predecessor of node n when doing so. The function $g(n)$ is the cost of the cheapest path to n found so far, regardless of the number of steps. The predecessors of a node n are denoted by n_-, the successors are denoted by n_+ and the cost of going from n to n' is $c_{n,n'}$.

Algorithm 1 has a time complexity of $O(L|E|)$. It is natural to require that $L \leq |N| - 1$, since no cheapest path can be longer than $|N| - 1$. Therefore, the time complexity of the algorithm can also be expressed as $O(|N||E|) \subseteq O(|N|^3)$. A complete solution to the **STR-ParetoLimited** problem is calculated for each node by Algorithm 1. Even if the cheapest path to the goal node has been found, and the length of the path is $\ll L$, the algorithm performs L outer iterations. This can lead to a large number of unnecessary calculations, especially if the number of nodes is large.

To remedy the long execution time, we have developed a new algorithm for the **STR-ParetoLimited** problem, presented in Section 5.2.

```
1  for each n ∈ N do g_0(n) ← +∞, p_0(n) ← nil
2  g_0(n_0) ← 0
3  for k = 1, ..., L do
4     for each n ∈ N do
5        g_k(n) ← g_{k−1}(n), p_k(n) ← p_{k−1}(n)
6     for each n ∈ N do
7        for each n' ∈ n_+ do
8           c ← g_{k−1}(n) + c_{n,n'}
9           if c < g_k(n') then
10             g_k(n') ← g(n), p_k(n') ← n
```

Figure 5.2: Algorithm 1 – for solving the AHOP problem, by Balakrishnan and Altinkemer [8].

5.2 A New Label-Correcting Algorithm

Our first new algorithm (Algorithm 2) is based on the AHOP version of the Bellman-Ford algorithm (Figure 5.2) [61], and just like that algorithm, it is a label-correcting algorithm. It improves the performance of Algorithm 1 as it is able to avoid a large percentage of the unnecessary calculations. Part of the improvement stems from a preprocessing step, involving the calculation of a tree consisting of paths to all nodes. Each such path has the least number of hops among those that have the lowest cost. Such paths correspond directly to solutions to the **STR-MinLengthMinCost** problem and will be called *Minimum Length Minimum Cost paths* (*MLMC-paths*). A tree of MLMC-paths from n_0 to all nodes in N is called an *MLMC-tree* and can be generated by Dijkstra's algorithm using compound path costs as previously described. The paths making up the MLMC-tree provide an upper bound on the length of the Pareto-optimal paths to each node in the graph. Calculations proceed in a manner similar to Algorithm 1, but the length of the MLMC-path to each node bounds the number of iterations for each node.

5.2.1 Preliminaries

To present the basic ideas of our algorithm and justify its correctness, we need the following definitions, which are illustrated by Table 5.1 for the graph in Figure 5.1.

A node n is called a *k-hop Pareto* node if there exists a Pareto-optimal path of length k from n_0 to n. We call such a path *k-hop Pareto-optimal*. For any k, the set of all k-hop Pareto nodes is denoted by N_k. As an example,

k	n_0	n_1	n_2	n_3	n_4	N_k	N_k^*
0	0^*	-	-	-	-	$\{n_0\}$	$\{n_0\}$
1	-	1^*	-	4	-	$\{n_1, n_3\}$	$\{n_1\}$
2	-	-	2^*	-	5	$\{n_2, n_4\}$	$\{n_2\}$
3	-	-	-	3^*	4^*	$\{n_3, n_4\}$	$\{n_3, n_4\}$

Table 5.1: Costs of k-hop Pareto paths, and the sets N_k and N_k^*, for the graph in Figure 5.1. The costs corresponding to MLMC-paths are marked by asterisks.

n_4 is both a 2-hop and a 3-hop Pareto node, showing that a node may be a member of multiple N_k sets. Although n_4 can be reached in 4 hops with the minimal cost 4, it is not a 4-hop Pareto node as the resulting path is not Pareto-optimal. The costs for the Pareto-optimal paths are presented in Table 5.1. $N_0 = \{n_0\}$ as the start node is the only node reachable in zero steps. The N_k^* sets store the set of nodes that occur at exactly k hops from n_0 in the MLMC-tree. Let k^*_{max} denote the height of the MLMC-tree, yielding $0 \leq k \leq k^*_{max}$ for the N_k^* sets. No Pareto-optimal path can be longer than k^*_{max} as longer paths must be at least as expensive. The N_k^* sets partition N, and as $N_k^* \subseteq N_k$, any path of depth k in the MLMC-tree is k-hop Pareto-optimal. Intuitively, $N_0^* = N_0 = \{n_0\}$.

Though the MLMC-tree may not be unique, the partitioning does not depend on the choice of tree, since the shortest path (in terms of number of hops) is chosen if several paths to the same node have the same cost.

Our algorithm exploits the following properties of Pareto-optimal paths: any Pareto-optimal path of length k must consist of a sequence of nodes occurring in $N_0, N_1, \ldots, N_k$. That is, any non-empty Pareto-optimal path to a node n' must consist of an extension of a shorter Pareto-optimal path to a predecessor n.

Theorem 1. *Let $n' \in N_k$. Suppose that*

$$\pi' = n_0 \to \ldots \to n \to n'$$

is a k-hop Pareto path. Then $n \in N_{k-1}$, and the path π composed by the first $k-1$ hops of this path is a $(k-1)$-hop Pareto path from n_0 to n.

Proof. By the definition of k-hop Pareto-optimality of π', there is no path π'' from n_0 to n such that:

$$\mathit{length}(\pi'') < \mathit{length}(\pi) \quad \text{and} \quad \mathit{cost}(\pi'') \leq \mathit{cost}(\pi)$$

k (path length)	g_k (cost)	p_k (predecessor)
1	4	n_0
3	3	n_2

Table 5.2: Reachability records for node n_3 after execution of Algorithm 2 on the graph in Figure 5.1.

or such that:

$$\mathit{length}(\pi'') = \mathit{length}(\pi) \quad \text{and} \quad \mathit{cost}(\pi'') < \mathit{cost}(\pi).$$

□

Theorem 1 implies that lines 7–10 of Algorithm 1 only have to be executed for nodes $n \in N_{k-1}$. The outgoing edges from other nodes cannot yield Pareto-optimal paths, nor can they result in updated node costs. The set N_{k-1} can be generated during iteration $k-1$ and used in the next iteration.

Our new algorithm uses the following identification of k-hop Pareto nodes: we can find a cheaper path to a node n using k hops than we could using $k-1$ hops only if $n \in N_k$. That is, only if there is a Pareto-optimal path to n using k hops.

$$n \in N_k \quad \Longleftrightarrow \quad g_{k-1}(n) > g_k(n). \tag{5.1}$$

Since an MLMC-tree is available, execution of lines 7–10 in Algorithm 1 can be skipped for the nodes $n' \in N_k^*$, as they are known to be k-hop Pareto nodes, and the costs $g_k(n')$ of the corresponding k-hop Pareto-optimal paths are already available in the MLMC-tree. Both of these facts are used to limit the number of calculations in our new algorithm.

The difference between the two algorithms is illustrated in Table 5.1, which shows all the values of $g_k(n)$ produced by our new algorithm. In contrast, Algorithm 1 produces the values of $g_k(n)$ for every node n and for each $k = 1, \ldots, 4$. Furthermore, it makes the same calculations for producing identical values, for instance, $g_1(n_3) = g_2(n_3) = 4$ and $g_3(n_3) = g_4(n_3) = 3$. For the same node, our algorithm calculates $g_1(n_3)$ only, and $g_3(n_3)$ is provided by the MLMC-tree.

5.2.2 Algorithm Details

The algorithm incrementally generates and updates a set of *reachability records* for each node. This set can be represented as one table for each node,

as seen in Table 5.2, which displays the reachability records for the node n_3 in the graph in Figure 5.1. Each reachability record $\langle k, g_k, p_k \rangle$ associated with a node indicates that the node can be reached from the start node n_0 in k hops at a cost of g_k using the predecessor p_k. A complete path from n_0 to n can always be reconstructed by considering the reachability record of the predecessor p_k for $k-1$ hops and continuing recursively until n_0 is reached. As the reachability records are traversed from n to n_0, the path needs to be reversed after retrieval. In our application, the number of hops corresponds to the number of agents required to realize the chain.

The first row of the table corresponds to using the minimal number of hops and the following rows correspond to using an increasingly larger number of hops, giving a path with progressively lower cost, until the least-cost path with the largest number of hops, which is found on the last row. For any number of hops smaller than the smallest k in the table, no chain can be found. For example, Table 5.2 shows that no chain of length $k = 0$ could be found, regardless of cost. Other "missing" values of k indicate that even if chains of length k exist, they are not Pareto-optimal. For example, the fact that Table 5.2 contains no record for $k = 2$ means that a chain of length 2 would be at least as expensive as a chain of length 1, the nearest smaller value of k. Thus, using a chain of length 2 would be pointless. However, using a chain of length 3 could be a useful alternative, as this would reduce the cost to 3 (in other words, increase the quality of the chain).

After termination, each reachability record is guaranteed to correspond to a Pareto-optimal path. Conversely, if there is a Pareto-optimal path of length l between n_0 and n, then n is guaranteed to have a reachability record for $k = l$. Thus, the fact that a target node τ_1 is associated with a reachability record $\langle k, g_k, p_k \rangle$ corresponds exactly to the existence of a Pareto-optimal relay chain between n_0 and τ_1 of length k and cost g_k, allowing τ_1 to be reached from the base station node using $k-1$ intermediate UAVs, of which one is the surveillance UAV.

Preprocessing. Our algorithm uses a preprocessing step (line 0) consisting of calculating the MLMC-tree, using for example Dijkstra's algorithm. In the tree, the chain to each node is the longest, and thus cheapest, to each node in the graph. We then retrieve the height k^*_{max} of this tree. This limits the number of relays required for any optimal relay chain for this graph, and also the depth to which the graph needs to be searched. For all k, we also extract the set N^*_k of all nodes of depth k in the tree.

For all nodes, we create an initial reachability record corresponding to the cheapest Pareto-optimal relay chain for each node. This allows us to abort execution individually for each node, as no cheaper path can be found

```
0  Calculate MLMC-tree and extract k*_max and all N*_k
1  for each n ∈ N \ {n_0} do g(n) ← +∞
2  for each n ∈ n_0− do                  // Incoming edges...
3     E ← E \ {(n, n_0)}                 // ... are removed
4  N_0 ← {n_0}
5  for k = 1, ..., min{M + 1, k*_max − 1} do
6     for each n' ∈ N*_k do
7        for each n ∈ n'_− do             // Incoming edges...
8           E ← E \ {(n, n')}            // ... are removed
9     N_k ← N*_k
10    for each n ∈ N_{k−1} do
11       for each n' ∈ n_+ do
12          c ← g_{k−1}(n) + c_{n,n'}    // To n' through n in k hops
13          if c < g(n') then
14             g(n') ← c                 // Lowest cost so far
15             g_k(n') ← c               // Lowest cost in k hops
16             p_k(n') ← n               // Predecessor for k hops
17             N_k ← N_k ∪ {n'}
```

Figure 5.3: Algorithm 2 – MLMC-tree-based label-correcting algorithm.

for lengths longer than the length of the MLMC-path. In Table 5.2, the initial record corresponds to the row where $k = 3$, and signifies that no path longer than three steps can decrease the path cost.

Initialization. Initially, $g(n) = \infty$ is set for all nodes. Then the algorithm constructs and uses a sequence of sets N_k, each of which is characterized, according to Theorem 1, by the fact that any k-hop Pareto-optimal path must consist of a $(k-1)$-hop Pareto-optimal path to a node $n \in N_{k-1}$ followed by a single outgoing edge from n.

Lines 2–4 provide initial values for the initial node n_0. Since $N_0^* = \{n_0\}$, a reachability record with $g_0(n_0) = 0$ must have been created during the preprocessing step, which indicates that it can be reached with cost 0 in 0 hops. Since the value $g(n)$ where $n \in N_k^*$ is not used in iteration k or any subsequent iteration, no value of $g(n_0)$ is assigned. Clearly no chain ending in n_0 can improve the value $g_0(n_0) = 0$, so all incoming edges to n_0 can be removed (lines 2–3). Finally, line 4 prepares for the first iteration by setting $N_0 = \{n_0\}$, indicating that any 1-hop Pareto-optimal path must consist of a path to n_0 in 0 hops (the empty path) followed by a single outgoing edge. This line handles all paths of length 0.

Main Part. The longest and thus cheapest Pareto-optimal path to each node was found during the preprocessing, and the main part of the algorithm begins by calculating the shortest (and thus most expensive) Pareto-optimal path. Then paths are found in order of increasing path length and decreasing cost.

Each iteration of lines 5–17, considers paths of length $k \geq 1$, up to a maximum of $\min\{M+1, k^*_{max}-1\}$. The upper bound reflects the fact that: (i) we allow at most M hops (UAVs), yielding a total path length of up to $M+1$ when edges to the base station and target are included, (ii) no paths of length greater than k^*_{max} can be Pareto-optimal, due to the meaning of k^*_{max}, and (iii) any Pareto-optimal path of length exactly k^*_{max} was already generated during the preprocessing.

Recall that any node $n' \in N^*_k$ occurs at depth k in the MLMC-tree. Reachability records for strictly shorter paths to n' were already created in earlier iterations, and a record for a path of length k was created during the preprocessing. Records for paths of length strictly greater than k cannot be created, as this would correspond to finding a path, which is longer but strictly cheaper than the one of length k, which was by definition a cheapest path. Thus, no new paths ending in any such n' can be Pareto-optimal, and we can remove all incoming edges to n' without affecting correctness or optimality (lines 6–8). However, we need to consider longer paths going *through* these nodes (line 9, explained further below).

In lines 10–17, we consider all potentially Pareto-optimal relay chains of length k. Any such chain must consist of a path to a node $n \in N_k$ followed by a single outgoing edge from n. We consider each such path in turn, determining its destination n' and calculating its cost c (lines 10–12). If the path has lower cost than the cheapest path found previously (with a cost of $g(n')$) we create a new reachability record with $g_k(n')$ set to the new path cost and $p_k(n')$ set to the new predecessor n (lines 13–16). Note that though reachability records are sparse, we know that any node in the set N_{k-1} does have a reachability record for $k-1$ due to the construction of such sets.

What remains is to prepare for the next iteration by constructing N_k according to its definition and characterization (relation 5.1). That is, we should construct N_k in such a way that any $(k+1)$-hop Pareto-optimal relay chain necessarily consists of a path of length k to a node $n \in N_k$ followed by a single outgoing edge from n. It is clear that no Pareto-optimal chain would use k hops to reach a node n if it could be reached in $k-1$ hops without incurring additional costs. This is stated in relation (5.1) which also covers the cases when n cannot be reached at all with paths of length $k-1$. Thus, it is only necessary to consider nodes whose costs were decreased by allowing paths of length k. This is achieved in line 9, for nodes in N^*_k

where we found a cheapest path of length k during the preprocessing, and in line 17, for nodes where we found a path of length k and of lower cost in this iteration.

For each node, Algorithm 2 produces a complete set of Pareto-optimal paths. In the preprocessing stage, when the MLMC-tree is created, it generates the longest and cheapest Pareto-optimal path, which is a path of minimum length among the paths of minimum cost. It then finds the shortest and most expensive Pareto-optimal path, and continues in order of increasing length and decreasing cost.

5.2.3 Correctness Proof

To prove that the variables and sets calculated by Algorithm 2 are correct, we use the notation presented in Section 5.2.1.

Theorem 2. *After iteration k of Algorithm 2, the calculated N_k sets defines the set of all k-hop Pareto nodes. For each $n' \in N_k$, the variable $g_k(n')$ defines the cost of a k-hop Pareto-optimal path from n_0 to n' and $p_k(n')$ defines the predecessor of n' in this path. For each $n' \in N$, $g(n')$ defines the cost of a cheapest path using at most k hops from n_0 to n' such that $n' \in N^*_{k'}$ with $k' > k$.*

Proof. In order to distinguish $N_k, g(n'), g_k(n')$ and $p_k(n')$ from the sets and variables generated by Algorithm 2, we will use the notation $N^A_k, g^A(n')$, $g^A_k(n'), p^A_k(n')$ for those that are created by Algorithm 2. The claims in the theorem can be formulated as follows:

1. $N^A_k = N_k$,
2. $g^A_k(n') = g_k(n'), \forall n' \in N_k$,
3. $p^A_k(n') = p_k(n'), \forall n' \in N_k$,
4. $g^A(n') = g_k(n'), \forall n' \in N$ such that $n' \in N^*_{k'}$, with $k' > k$.

We will prove by induction that the claims in the theorem hold for $k = 0, \ldots, \min\{M+1, k^*_{max} - 1\}$. All line numbers refer to the pseudocode in Figure 5.3.

The claims 1–4 hold for $k = 0$ as the start node n_0 is the only node that is reachable in zero steps and the cost of such a path is $g_0(n_0) = 0$ with $p_0(n_0) = nil$. As only n_0 is reachable in $k = 0$ steps, $N_0 = N^*_0 = \{n_0\}$ applies. For all other nodes $n' \in N, g(n') = \infty$. In Algorithm 2, the cost $g^A_0(n_0) = 0$ and the predecessor $p^A_k(n_0)$ are assigned during execution of the

MLMC-tree. All N_k^* sets are extracted after the execution of the MLMC-tree (line 0). As n_0 is the only node reachable in zero step, $N_0^A = N_0^* = \{n_0\}$ must apply. The cost $g(n') = \infty, \forall n' \in N \setminus \{n_0\}$ is set in line 2.

Assume that the claims hold for some $k = i < \min\{M+1, k^*_{max} - 1\}$.

We will show that the claims 1–4 apply for $k = i+1$. We begin by proving that N_{i+1}^A correctly defines the set of all $(i+1)$-hop Pareto nodes. Line 9 implies that if $n' \in N_{i+1}^*$, then $n' \in N_{i+1}^A$. To prove that N_{i+1}^A correctly defines the set of all $(i+1)$-hop Pareto-nodes, we must prove that i) for all $n' \in N_{i+1}^A \setminus N_{i+1}^*$, n' is an $(i+1)$-hop Pareto-node, and ii) if $n' \notin N_{i+1}^A$, then n' is not an $(i+1)$-hop Pareto-node. We begin by proving the former.

For node $n' \in N_{i+1}^A \setminus N_{i+1}^*$, the existence of an $(i+1)$-hop path from n_0 to n' with a cost of less than the cost of a cheapest path to n' with at most i hops, is equivalent to n' being an $(i+1)$-hop Pareto node, by (5.1). By the induction hypothesis, at the end of iteration $k = i$, the value of $g^A(n')$ is the cost of the cheapest path using at most i hops. The case when n' is added to N_{i+1}^A on line 17 corresponds to the case of finding an $(i+1)$-hop path with a cost less than $g^A(n')$. Thus n' is an $(i+1)$-hop Pareto node according to (5.1).

To prove ii), consider $n' \notin N_{i+1}^A$. By i), $n' \notin N_{i+1}^*$. Assume by contrary that $n' \in N_{i+1} \setminus N_{i+1}^*$. By Theorem 1 and (5.1) there should exist some $n \in N_i$ such that $g_i(n') > g_{i+1}(n) + c_{n,n'}$. By claims 1 and 4, $n \in N_i^A$. Then $c < g(n')$ in line 13 must hold at least once. This implies that $n' \in N_{i+1}^A$, which contradicts our assumption that $n' \in N_{i+1} \setminus N_{i+1}^*$. Thus ii) holds.

For any $n' \in N_{i+1}^A$, the for-loop in lines 10–17 assures that the equality

$$g_{i+1}^A(n') = \min_{n \in n'_- \cap N_i} \{g_i^A(n) + c_{n,n'}\}.$$

holds. The right hand side is equal to $g_{i+1}(n')$ by its definition. Thus claim 2 holds for $k = i+1$. As any change in $g_{i+1}^A(n')$ is associated with a change in $p_{i+1}^A(n')$, claim 3 also holds.

To prove claim 4, we consider the two cases iii) $n' \in N_{i+1} \setminus N_{i+1}^*$ and iv) $n' \notin N_{i+1}$. Note that claim 4 does not refer to $n' \in N_{i+1}$. For case iii), lines 14–15 implies that $g^A(n')$ changes at iteration $i+1$ only if $g_{i+1}^A(n')$ changes, and if $g_{i+1}^A(n')$ changes at iteration $i+1$, then $g^A(n') = g_{i+1}^A(n')$. Then as claims 1 and 2 were earlier shown to hold for $k = i+1$, claim 4 holds for $n \in N_{i+1} \setminus N_{i+1}^*$. For case iv), $g^A(n')$ has the same value at the end of iterations i and $i+1$. Also, $g_i(n') = g_{i+1}(n')$. Assuming that $n' \in N_k^*$ with $k' > i+1$, then by claim 4 for $i = k$, the value of $g^A(n')$ is equal to $g_i(n')$ at the end of iteration i. The same holds for $g^A(n')$ at the end of iteration $i+1$. Thus claim 4 applies for both case iii) and iv). This completes the proof for $k = i+1$.

As the claims hold $k = 0$, by induction they hold for $k = 0, \ldots, \min\{M + 1, k^*_{max} - 1\}$ as this is the exact number of performed iterations. □

See [17, 15, 16] for additional details and proofs.

5.2.4 Time Complexity

The label-correcting algorithm presented here performs a calculation of the MLMC-tree, which can be performed in $O(|E| + |N| \log |N|)$ time using Dijkstra's algorithm (line 0). Clearing the costs on line 1 takes $O(|N|)$ time, and lines 2–4 requires at most $|E|$ time and the time complexity for the preprocessing is $O(|E| + |N| \log |N|)$. At most $k^*_{max} - 1$ outer iterations are performed, where each iteration will goes through $|E| \leq |N|^2$ edges and the time complexity of the main part of the algorithm is $O((k^*_{max} - 1)|E|) \subseteq O((k^*_{max} - 1)|N|^2)$. Thus the time complexity for the complete algorithm is $O(k^*_{max}|N|^2) \subseteq O(|N|^3)$. The time complexity is the same as for the original Bellman-Ford, but typically, $k^*_{max} \ll |N|$ applies for our problems. Also, $|E| \ll |N|^2$ often applies as each node is only connected to a small subset of nodes.

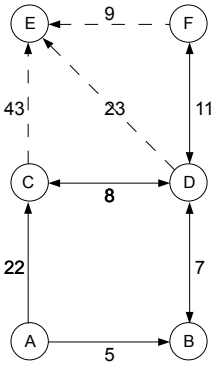
5.2.5 Improved Preprocessing

As can be seen on line 6 in Figure 5.3, the number of main iterations is limited by $\min(M+1, k^*_{max}-1)$. From that, it is obvious that no unnecessary iterations are performed in the main part of Algorithm 2.

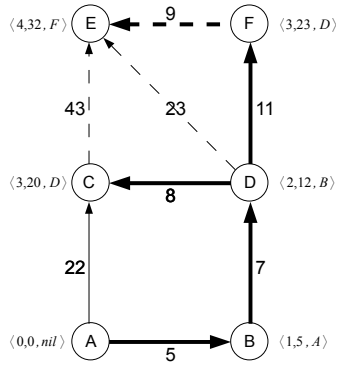
However, if the graph consists of a very large number of nodes and the graph topology is such that $k^*_{max} \gg M + 1$, then some unnecessary calculations will be performed during the preprocessing step, for nodes that will not be reached in the main part of the algorithm. Iterations for such nodes can be removed by introducing a depth-limit in the calculation of the cheapest path tree. When the depth of a node reaches $M+1$, the node cannot be part of a relay chain unless the node is the target node. This fact is exploited on line 5 in Algorithm 2, and no calculations starting in the node will be performed. As the value of k^*_{max} is retrieved *after* the calculation of the MLMC-tree, a depth limit of $M + 1$ is the strongest depth limit of the tree calculation that can be used without affecting the algorithm's properties.

5.2.6 Example

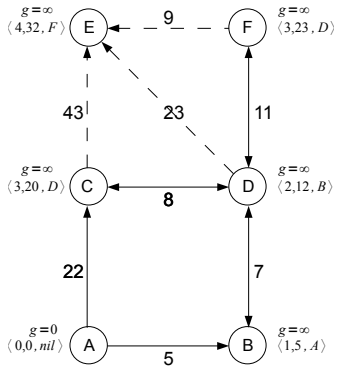
We will now show how the algorithm works in a small example, displayed in Figure 5.4a, with $n_0 = A$, $\tau_1 = E$ and $M = \infty$. Communication edges are solid and surveillance edges are dashed.



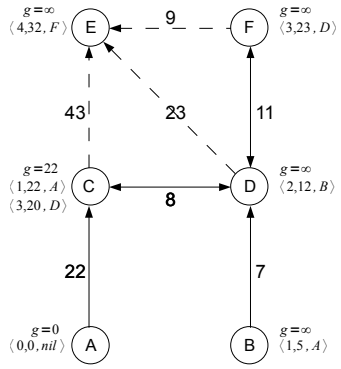
(a) The initial graph.



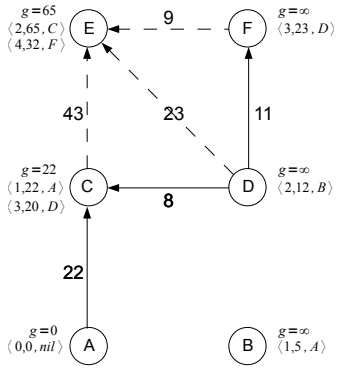
(b) MLMC-tree calculated during preprocessing.



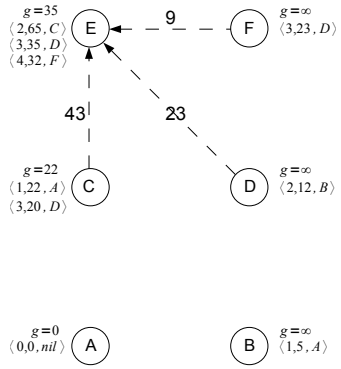
(c) Before the first iteration, with node costs set.



(d) After the first iteration.



(e) After the second iteration.



(f) After the third iteration.

Figure 5.4: Example showing the execution of the label-correcting algorithm.

Length	Cost	Chain
2	65	$A \to C \to E$
3	35	$A \to B \to D \to E$
4	32	$A \to B \to D \to F \to E$

Table 5.3: The chains to node E for the graph in Figure 5.4.

During the preprocessing, the MLMC-tree marked by the heavy edges in Figure 5.4b is calculated. The reachability records $\langle hops, cost, predecessor \rangle$ are displayed next to each node.

For node E, the chain is $A \to B \to D \to F \to E$. From the MLMC-tree, the N_k^* sets are determined: $N_0^* = \{A\}, N_1^* = \{B\}, N_2^* = \{D\}, N_3^* = \{C, F\}, N_4^* = \{E\}$. As $k_{max}^* = 4$ the number of iterations is limited to 3. Initially, the node cost g is set for all nodes, $g(A) = 0$, and $g(n) = \infty$ for all other nodes, and any incoming edges to the start node are removed. The graph after these modifications is displayed in Figure 5.4c. $N_0 = N_0^* = \{A\}$ is set before the first iteration.

At the start of the first iteration, all incoming edges to the nodes in $N_1^* = \{B\}$ are removed. All outgoing edges of the nodes in $N_0 = \{A\}$ are treated. A new chain of length 1 and cost 22 is found to node C. As $g(C) = \infty$, the new chain is stored and $g(C)$ is updated (Figure 5.4d). The set of nodes for treatment in the next iteration, N_1, is constructed during this iteration and consists of the nodes B and C. $N_1 = \{B, C\}$.

In the second iteration, all incoming edges to the nodes $N_2^* = \{D\}$ are removed. Next, the outgoing edges of nodes in $N_1 = \{B, C\}$ are treated, and a chain of length 2 and cost 65 to node E is found and stored: $A \to C \to E$. As the edge from node B to D was removed, no new chain involving node B can be found. $N_2 = \{D, E\}$. Figure 5.4e displays the graph at this stage.

Iteration 3 is the final iteration as $k_{max}^* = 4$. Incoming edges to $N_3^* = \{C, F\}$ are removed. All nodes in $N_2 = \{D, E\}$ are treated and a new chain of length 3 and cost 35 to node E is found: $A \to B \to D \to E$. Node E has no outgoing edges, thus no new chain from it can be found. $N_4 = \{E\}$. The final graph is displayed in Figure 5.4f. All chains to node E and their costs are displayed in Table 5.3.

5.3 A New Dual Ascent Algorithm

The **STR-MinCostLimited** problem corresponds directly to the shortest path problem with hop-constraints [31], where "shortest" in this case

means "lowest cost". This problem does not require finding all chains, only a sufficiently short chain. To solve the problem, label-correcting algorithms such as Algorithm 1 are often used, and a single sufficiently short chain is extracted from the solution. However, doing so requires calculating a potentially large set of relay chains, and discarding all but one. As Algorithm 2 performs fewer calculations and should have a shorter execution time, we limit our discussion to that algorithm. During preprocessing, the longest chain is calculated, and then the shortest and most expensive chain is calculated in the first iteration. In later iterations, progressively longer and less expensive chains are calculated. Thus, if the longest chain has only marginally too many hops, calculations must continue until all shorter chains have been calculated. There is no advantage to the fact that the first chain was "almost" feasible.

We therefore present another algorithm (Algorithm 3) that begins with calculating the cheapest chain and then calculates more expensive chains, which are shorter. This type of algorithm is based on the dual ascent approach to optimization. From a mathematical perspective, it tries to maximize the piece-wise affine Lagrangian dual function by traversing a dual segment at a time.

As mentioned previously, common shortest path algorithms such as Dijkstra's cannot be used to calculate a complete *set* of relay chains directly. However, such algorithms can be used to calculate a single relay chain, provided that one exists. Any shorter chain is more expensive, and for a cheapest path algorithm to find such a chain, the edge costs must be increased. If all edge costs are increased by an equal amount, the cost of a longer path is increased proportionally more than a shorter path. If all edge costs are increased by ϵ, a chain of length k has its cost increased by $k\epsilon$. The cost of another relay chain, of length $m > k$, is increased by $m\epsilon$. Thus, the cost of the longer chain is increased by $(m-k)\epsilon$ more than the cost of the shorter chain. This is the basic idea behind the dual ascent algorithm, which repeatedly calculates a MLMC-tree and an edge cost increase ϵ. As all edge costs are increased by ϵ, progressively shorter paths are found as the costs of longer chains are increased more than the costs of shorter chains.

5.3.1 Algorithm Details

The pseudocode for the dual ascent algorithm is presented in Figure 5.5.

The parameter l is the number of acceptable hops and is initialized to $M+1$, with the one added for the last hop between the surveillance UAV and the target. The parameter α, representing the extra cost that is added to each edge, is initialized to α_0, which is commonly set to 0 to ensure that

1 $\alpha \leftarrow \alpha_0, l \leftarrow M + 1$
2 **loop**
3 Calculate MLMC-tree from n_0 using costs $c'_{n,n'} = c_{n,n'} + \alpha$.
From the tree, obtain π and y_n, q_n for all $n \in N$
4 **if** *length*$(\pi) \leq l$ **then return** π *// Feasible solution*
5 $H \leftarrow \{(n, n') \in E : q_{n'} \geq q_n + 2\}$
6 **if** $H = \emptyset$ **then return** failure *// No chain can be shortened*
7 **for** $\{n, n'\} \in H$ **do**
8 $\epsilon_{n,n'} \leftarrow \frac{c'_{n,n'} + y_n - y_{n'}}{q_{n'} - q_n - 1}$
9 $\epsilon \leftarrow \min \epsilon_{n,n'}$
10 $\alpha \leftarrow \alpha + \epsilon$

Figure 5.5: Algorithm 3 – dual ascent algorithm.

all paths are found (line 1). Other values of α_0 are possible, but at the risk of decreasing the path length too much, thus finding a more expensive chain than required. The chain between n_0 and τ_1 is calculated through repeated calculations of the MLMC-tree, using modified edge costs $c'_{n,n'} = c_{n,n'} + \alpha$ (line 3). From the MLMC-tree, the chain π from n_0 to τ_1 is retrieved, together with the depth q_n and the optimal chain cost y_n, for all nodes. The depth q_n is the number of hops required to reach node n from n_0 in the MLMC-tree, and the optimal path cost y_n is the minimum cost required to reach node n from n_0 given this value of α (line 3).

If the chain between n_0 and τ_1 is short enough, i.e. the number of hops is less than l, then a valid solution has been found and the algorithm terminates (line 4).

The next step in the algorithm (line 5) is to find the edges that could potentially be included in the MLMC-tree in order to decrease the length of a chain. The condition for an edge that could be added to the MLMC-tree is that the number of hops required to reach the edge end node $q_{n'}$ is at least two more than the number of hops required to reach the edge start node q_n. If the difference in the number of hops was one, the edge could already be included in the tree and would not make any path in the tree shorter, thus the difference in depth must be at least 2. The set H consists of all edges with a depth difference of ≥ 2 between the nodes. If $H = \emptyset$ then we already have a tree of minimum depth and no edge can be included in order to decrease the length of any path in the tree, and the algorithm terminates (line 6).

An iteration through H is performed (lines 7–8), with the intention to calculate the edge cost increase that should be applied to all edge costs in

order to decrease the length of at least one path in the tree. Since we know that the edges in H are not already included in the tree, going through n to reach n' is more expensive than the current path to n, i.e. $y_n + c'_{n',n} > y_{n'}$ holds for all edges in H. Making the path to n' through n equally expensive requires increasing the cost by $y_n + c'_{n,n'} - y_{n'}$, *more* than the cost of the shorter path. The cost increase must be distributed over $q_{n'} - (q_n + 1) = q_{n'} - q_n - 1$ edges, yielding an increase per edge of $\frac{c'_{n,n'}+y_n-y_{n'}}{q_{n'}-q_n-1}$.

To ensure that no chain on the problem instance's convex hull (see Section 5.3.2) is missed when the edge cost increase is applied to all edge costs, the minimum $\epsilon_{n,n'}$ is used and it is added to the value of α, on lines 9–10. This will increase the cost of the shorter path by exactly the amount required to make the shorter path equally expensive as the longer path, and a shorter path will be preferred in such situation due to use of the compound cost (see page 41). This concludes one iteration of the algorithm, and the process of calculating the MLCM-tree is then repeated, using the new value of α (line 3).

The edge(s) yielding the minimal ϵ is normally the set of edges that will be included in the MLMC-tree and the set of edges ending in same nodes as the included edges will leave the tree. When several edges in H have the same end node, only one of the edges is included in the tree.

Although the dual ascent algorithm is intended to create solutions to the **STR-MinCostLimited** problem, it can be modified to solve the **STR-ParetoLimited** problem. For this problem, line 4 does not return the chain π, but rather it is yielded as a partial solution and execution continues until $H = \emptyset$.

5.3.2 Theoretical Properties

The dual ascent algorithm follows the problem instance's convex hull and finds all Pareto-optimal solutions on the hull. Figure 5.6 shows the Pareto-optimal relay chains marked by circles, and the convex hull of a particular problem. Any Pareto-optimal chain not on the convex hull of the problem instance, such as the relay chain marked by a square, will not be found. Thus, the algorithm is not optimal for the **STR-MinCostLimited** problem, as it occasionally will find a slightly shorter chain than the longest feasible. However, Pareto-optimal chains that are not on the convex hull can be found using a branch-and-bound post-processing step [18]. As the cheapest and the shortest chains are on the convex hull, they will always be found.

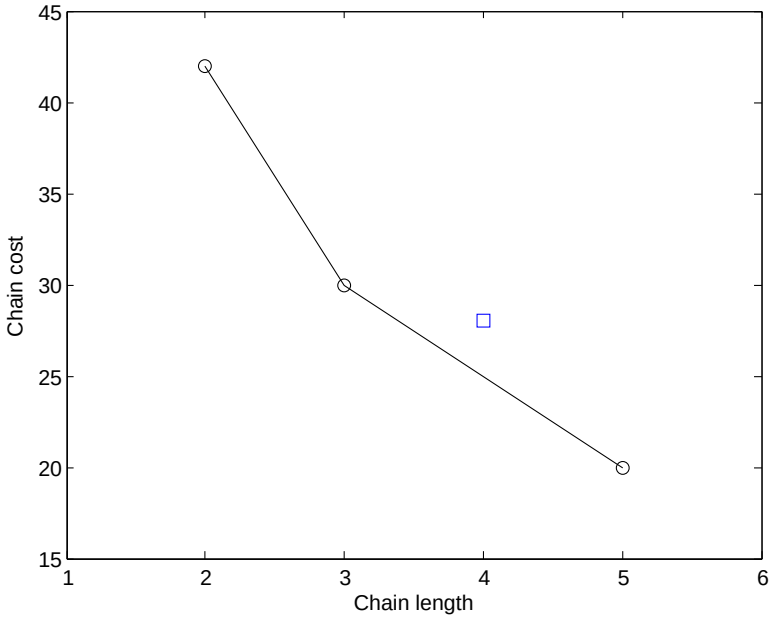


Figure 5.6: The convex hull of a problem instance is marked by a solid line and the Pareto-optimal relay chains on the hull are marked with circles. The square marks a Pareto-optimal chain that will not be found by the dual ascent algorithm.

Monotonicity With Respect to α. We will now prove that increasing the value of α yields shorter but more expensive chains.

Theorem 3. *Let $n \in N$ be an arbitrary node distinct from, but reachable from, the start node n_0. Consider the paths π_1 and π_2 generated from n_0 to n in the MLMC-tree for two distinct values of α, denoted by α_1 and α_2. If $0 \leq \alpha_1 < \alpha_2$, then $cost(\pi_1) \leq cost(\pi_2)$ and $length(\pi_1) \geq length(\pi_2)$. That is, increasing α cannot increase the length or decrease the true cost of an optimal path from n_0.*

Proof. Here we make use of two distinct path cost measures: the true cost of a path π is defined in terms of the communication and surveillance costs only, and is denoted by $cost(\pi)$. As the edge costs are incremented by α, we define the modified cost of π given a value of α to be $cost_{mod}(\pi, \alpha)$. Obviously, $cost_{mod}(\pi, \alpha) = cost(\pi) + \alpha \cdot length(\pi)$.

It is clear that $cost_{mod}(\pi_1, \alpha_1) \leq cost_{mod}(\pi_2, \alpha_1)$, since π_1 is a cheapest path for $\alpha = \alpha_1$. Analogously, $cost_{mod}(\pi_2, \alpha_2) \leq cost_{mod}(\pi_1, \alpha_2)$ must apply.

Suppose that $cost(\pi_2) < cost(\pi_1)$, so that using a larger α resulted in a

decrease in true path cost. If the paths were of equal length, changing α would always change the modified costs of π_2 and π_1 by the same amount, and π_2 would have been strictly preferred over π_1 for all values of α. As π_1 was returned for $\alpha = \alpha_1$, this cannot be the case. Suppose instead that $\mathit{length}(\pi_2) > \mathit{length}(\pi_1)$. Then the modified cost of π_2 would increase by a greater amount than the modified cost of π_1 when α is increased. Thus, if π_1 was returned for a smaller value of α, π_2 cannot be preferred for a larger value of α. The case where $\mathit{length}(\pi_2) < \mathit{length}(\pi_1)$ remains. But if π_2 is shorter and has a lower true cost, then its modified cost must be less for any positive value of α, and we cannot have $\mathit{cost}_{mod}(\pi_1, \alpha_1) \leq \mathit{cost}_{mod}(\pi_2, \alpha_1)$. Thus, it must be the case that $\mathit{cost}(\pi_2) \geq \mathit{cost}(\pi_1)$.

Suppose that $\mathit{length}(\pi_2) > \mathit{length}(\pi_1)$, so that using a larger α resulted in an increase in path length. We know that $\mathit{cost}_{mod}(\pi_1, \alpha_1) \leq \mathit{cost}_{mod}(\pi_2, \alpha_1)$. When α is increased from α_1 to α_2, the modified cost of π_2 must increase by a strictly greater amount than the modified cost of π_1, since π_2 is longer and modified costs increase in proportion to length. Thus, $\mathit{cost}_{mod}(\pi_1, \alpha_2) < \mathit{cost}_{mod}(\pi_2, \alpha_2)$, which contradicts $\mathit{cost}_{mod}(\pi_2, \alpha_2) \leq \mathit{cost}_{mod}(\pi_1, \alpha_2)$. Thus $\mathit{length}(\pi_2) \leq \mathit{length}(\pi_1)$. □

Convergence and Time Complexity. Let T_D denote the tree depth of the MLMC-tree, calculated as $T_D = \sum_{n \in N} q_n$. T_D is bounded from below by $|N|-1$ as the tree with the least depth is star shaped with one node in the middle having depth 0 and the other $|N|-1$ nodes having depth 1, yielding $T_D = |N| - 1$. The tree with the maximum depth has two nodes with degree one, and all other nodes have degree two, i.e. a single path. In that case, the first node has $q_n = 0$, the second node $q_n = 1$ and so on, yielding $T_D = \frac{|N|(|N|-1)}{2}$. As this is the maximum possible depth we have $|N| - 1 \leq T_D \leq \frac{|N|(|N|-1)}{2}$. From this it is apparent that the number of possible iterations is bounded by $|N|^2$. In each iteration, each of the $|E| \leq |N|^2$ edges is checked for inclusion in the MLMC-tree, and the MLMC-tree is calculated. Such a tree can be computed in $O(|E| + |N| \log |N|)$ time. Therefore, the maximum time complexity is $O(|N|^2(|E| + |E| + |N| \log |N|)) \subseteq O(|N|^4)$. In most cases, this is a severe overestimation as graphs requiring $|N|^2$ iterations are not common on the relay problems.

From Figure 5.5, it is evident that only edges with $q_{n'} \geq q_n + 2$ can enter the MLMC-tree. When such an edge is included in the MLMC-tree, it will decrease the depth of node n' and all nodes in the subtree rooted in n'. As the depth will not be increased for any node, T_D will decrease. As T_D is decreased by at least one in every iteration, the algorithm converges.

For additional proofs, the reader is referred to [18, 19].

5.3.3 Example

We will now show how the dual ascent algorithm works for finding a chain from $n_0 = A$ to $\tau_1 = E$ with a length of at most three hops. The start value $\alpha_0 = 0$ and the initial graph is displayed in Figure 5.7a.

In iteration 1, the MLMC-tree-is calculated and the cheapest chain, $\pi_1 = A \rightarrow B \rightarrow D \rightarrow F \rightarrow E$, $cost(\pi_1) = 32$ and $length(\pi_1) = 4$, is found (Figure 5.7b). The set $H = \{(A, C), (D, E)\}$. Calculating ϵ for (A, C) with the variables $c_{A,C} = 22, y_A = 0, y_C = 20, q_A = 0, q_C = 3$ yielding $\epsilon_{A,C} = \frac{22+0-20}{3-0-1} = 1$. Analogously, the edge (D, E) yields $\epsilon_{D,E} = \frac{23+12-32}{4-2-1} = 3$. The cost increase $\alpha = \epsilon_{A,C} = 1$ is applied to all edge costs.

In the second iteration, the new edge costs are used when the MLMC-tree is calculated. π_1 is still the cheapest chain ($cost_{mod}(\pi_1) = 36$) and the new MLMC-tree is displayed in Figure 5.7c. The edge (A, C) enters the MLMC-tree and the edge (D, C) leaves it. The length of the chain to C is decreased, but the chain to E is not affected. In this iteration, $H = \{(C, E), (D, E)\}$, which yields $\epsilon_{C,E} = \frac{44+23-36}{4-1-1} = 15.5$ and $\epsilon_{D,E} = \frac{24+14-36}{4-2-1} = 2$

All edge costs are increased by $\epsilon = \epsilon_{D,E} = 2$ before calculating the new MLMC-tree in iteration 3 (Figure 5.7d). A new chain $\pi_2 = A \rightarrow B \rightarrow D \rightarrow E$ is found. As $cost_{mod}(\pi_1) = cost_{mod}(\pi_2) = 44$ and $length(\pi_2) = 3$, π_2 will be preferred as it requires fewer hops. The chain π_2 is sufficiently short and the algorithm terminates.

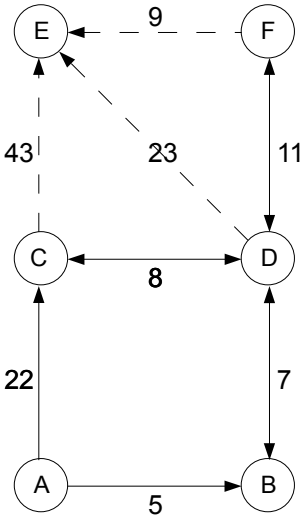
5.3.4 Performance Improvements

The basic dual ascent algorithm as displayed in Figure 5.5 allows for several performance optimizations, which have been used in our empirical testing.

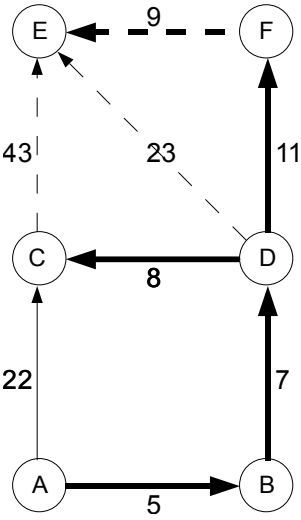
Optimized Generation of MLMC-trees. For all iterations except the first, the generation of an MLMC-tree can be optimized by making use of the fact that an MLMC-tree has already been calculated, albeit with different modified edge costs.

First, the cost of each edge in the previously created tree is increased by the recently calculated ϵ, and the cost of each node n is increased by $q_n\alpha$ (that is, the node cost is increased in proportion to its depth). This ensures that the cost y_n of any node n correctly reflects the new modified cost of the path from n_0 to n.

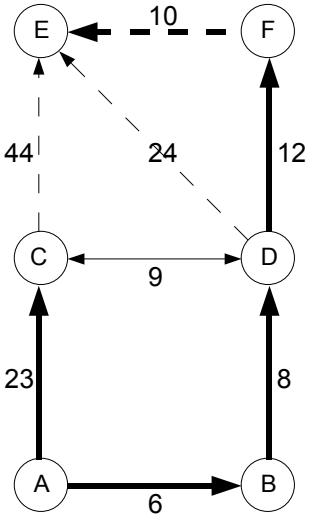
After the increase in edge costs, the tree is no longer an MLMC-tree. For some nodes, it is possible to find cheaper paths in the graph than those that are currently included in the tree. Any such paths must include at least one of the edges that yielded the current value of ϵ. We can therefore begin "repairing" the tree starting at the source nodes of those edges, rather than



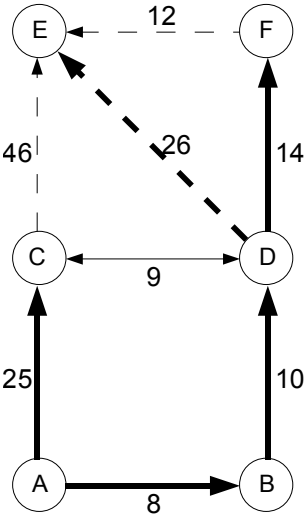
(a) Initial graph.



(b) The MLMC-tree calculated in iteration 1 is marked by heavy edges.



(c) In iteration 2, a new MLMC-tree is calculated after $\alpha = 1$ has been added to all edge costs.



(d) A sufficiently short chain is found in iteration 3.

Figure 5.7: The dual ascent algorithm is applied to finding a chain from A to E of at most three hops.

from the root node n_0, by initializing Dijkstra's algorithm with a priority queue containing exactly those source nodes. As the tree is traversed in the standard manner, only those nodes to which we find a cheaper path than in the previous calculation of the tree are added to the priority queue. Thus, parts of the tree where the chain is already optimal will not be visited, speeding up the calculation of the tree. This way of starting from an existing tree is sometimes referred to as Ford's algorithm [56].

Search Space Limitation. As seen in the example in Section 5.3.3, the dual ascent algorithm decreases T_D in each iteration, but the chain between n_0 and τ_1 might not be affected. Depending on the difference in edge costs and which edge determines $\epsilon_{n,n'}$, there can be many iterations that do not affect the chain between n_0 and τ_1. However, to increase the chance that the relevant chain is affected, the *search space* can be limited. To guarantee that no relevant chains are missed, the desired effect is to remove the parts of the graph that are irrelevant for the current target.

An example of such irrelevant nodes are the nodes that require too many steps from n_0. Such nodes can be removed, and to determine the number of steps required to reach the node from n_0, a *shortest path tree* starting in n_0 can be used. This tree is calculated by using for example Dijkstra's algorithm and setting all edge costs to 1. The output of such a calculation is the minimum number of hops required to reach each node, denoted by q_n^{SP} $\forall n \in N$. After the shortest path tree has been calculated, all nodes not fulfilling $q_n^{SP} \leq M + 1$ are removed from N, as they require too many hops to be reached. Naturally, if it is not possible to reach a node using the maximum number of steps allowed, it cannot be part of a valid path between n_0 and τ_1 and it can be safely removed.

The same pruning of the search space can be performed by calculating another shortest path tree, this time rooted in τ_1. Let q_n^{SPt} denote the number of hops required to reach node n from node τ_1. After calculation of the shortest path tree rooted in τ_1, the value of q_n^{SPt} is set for all reachable nodes. As both q_n^{SP} and q_n^{SPt} are available, the condition of $q_n^{SP} + q_n^{SPt} \leq l$ can be applied. Nodes not satisfying this condition can be removed in order to increase the performance of the algorithm, without affecting the optimality or completeness properties. Obviously, if q_n^{SP} hops are required to reach node n from n_0, a valid path between n and τ_1 may require no more than $l - q_n^{SP}$ hops.

Limiting the search space in this way can potentially improve performance more for small values of M, as larger parts of the search space are discarded. Thus, it is not certain that a lower value of M yields slower execution, instead the search space limitation may yield large improvements

when solving the **STR-MinCostLimited** problem.

This way of improving performance has also been suggested as a preprocessing step for the NP-hard problem of finding shortest paths fulfilling both a limit on the number of hops and a limit on the path cost [39].

5.4 Summary

Of the single target relay problems defined earlier, efficient algorithms exist for the **STR-MinLengthMinCost** and **STR-MinCostMinLength** problems. In this chapter, new algorithms for solving the **STR-MinCostLimited** and the **STR-ParetoLimited** problems have been presented. The **STR-ParetoLimited** problem is solved using a new label-correcting algorithm. The algorithm uses a preprocessing step consisting of calculating an MLMC-tree from the start node to all nodes in the graph. Each path in such a tree consists of the path having the fewest steps from the set of cheapest paths. The tree provides an upper bound on the path length to each node, and this is used to terminate calculations for each node in the main part of the algorithm. This avoids many unnecessary calculations, which improves the execution time significantly, as demonstrated in Chapter 7.

The **STR-MinCostLimited** problem can also be solved by the label-correcting algorithm, but as it only requires finding a sufficiently short path, other methods may be more efficient. This problem can be solved using a dual ascent algorithm, which repeatedly calculates an MLMC-tree. After each calculation of the tree, it is checked whether the path from the base station node n_0 to the target node τ_1 is sufficiently short. If so, the path is a feasible solution and the algorithm terminates. If not, the algorithm calculates and applies a cost increase to all edge costs. The cost increase is calculated in such a way that at least one path in the tree is shortened, and the process of repeatedly calculating the MLMC-tree and the edge cost increase is continued until a sufficiently short path is found or until no path in the tree can be shortened.

Chapter 6

Relay Positioning for Multiple Targets

We have previously presented algorithms for calculating relay chains for surveillance of a single target. In this chapter, we will discuss how simultaneous surveillance of several targets can be modeled. We also discuss algorithms for positioning relays in such cases.

Assume that there are several targets that must be surveilled. In such a case, one option is to use the algorithms already described to calculate several chains, creating one independent relay chain to each target. If the UAVs are only capable of relaying a single stream of information, then this is the best that we can do. However, if the UAVs can handle several streams of information, then we can take advantage of this and calculate a *relay tree* where some UAVs relay information from several surveillance UAVs. This synergy may allow us to decrease the number of UAVs required to surveil the targets. We will now go on and define problems for surveillance of multiple targets and then discuss algorithms for solving them.

6.1 Definition of the Multiple Target Relay Problems

For problems involving several targets, the same assumptions as for single targets are made (see Section 3.1), but instead of a single target, assume as given a set $T = \{t_1, \ldots, t_l\} \subseteq R^3 \setminus U$ of l surveillance target positions.

A *relay tree* between x_0 and the target positions $\{t_1, \ldots, t_l\}$ is a set of relay chains that together form a tree structure. As an example, consider Figure 1.2 on page 4, where there are two chains: $[x_0, x_1, x_2, x_3, x_5, x_6, t_1]$

and $[x_0, x_1, x_2, x_3, x_4, t_2]$. Note that these chains may share positions, corresponding to a UAV being used in multiple chains. For each target $t_i \in T$, there is a relay chain beginning in x_0 and ending in t_i. Let L be the number of UAVs required to realize the tree and let the non-target positions in the tree be denoted by $[(x_0, \ldots, x_L])$. In Figure 1.2, there are six such UAV positions: $x_1 - x_6$. Also, let x^- denote the unique predecessor of position x. For example, in Figure 1.2 the predecessor of x_5 is x_3: $x_5^- = x_3$. Then, the *cost* of a relay tree is

$$\sum_{i=1}^{L} c_{comm}(x_i^-, x_i) + \sum_{i=1}^{l} c_{surv}(t_i^-, t_i)$$

Let the *length* of a relay tree be the number of agents required to realize the chain, that is $L + 1$ for L UAVs and the base station.

With the necessary definitions in place, we define the following multiple target relay problems. Some of the targets assume a limit on the number of available UAVs, and just like in the single target case, we use the letter M to denote this limit. Setting $M = \infty$ requires finding all solutions, regardless of length.

MTR-MinLengthMinCost: Find a relay tree of minimum length among the trees of minimum cost. A solution to this problem is a tree s such that for all other trees t, $cost(s) \leq cost(t)$ and $cost(s) = cost(t) \rightarrow length(s) \leq length(t)$. This corresponds to using the highest quality tree that can be realized with access to an unlimited number of UAVs, with a preference for using fewer UAVs if this is possible without compromising quality.

MTR-MinCostMinLength: Find a relay tree of minimum cost among the trees of minimum length. A solution to this problem is a tree s such that for all other trees t, $length(s) \leq length(t)$ and $length(s) = length(t) \rightarrow cost(s) \leq cost(t)$. This is useful if minimizing the number of UAVs is strictly more important than maximizing quality.

MTR-MinCostLimited: Find a relay tree of minimal cost among the chains that use at most M UAVs. A solution to this problem is a tree s such that $length(s) \leq M + 1$ for all other trees t, and $length(t) \leq M + 1 \rightarrow cost(s) \leq cost(t)$. This corresponds to a desire to find the highest quality relay tree that can be realized within the given limit on the number of UAVs.

MTR-ParetoLimited: Find a set of *Pareto-optimal* relay trees that is complete up to a given upper limit on the number of available UAVs. A tree s is Pareto-optimal for up to M UAVs if $length(s) \leq M + 1$ and for all trees t of length at most $M + 1$, $length(t) < length(s) \rightarrow cost(t) > cost(s)$ and $cost(t) < cost(s) \rightarrow length(t) > length(s)$.

As we will see, these problems are difficult to solve quickly. Therefore, we will focus on finding algorithms suitable for calculating approximate solutions to the **MTR-MinLengthMinCost** and **MTR-MinCost-MinLength** problems in the rest of this thesis. Solving these problems requires minimization of the total cost or the length of the tree. This makes the multiple target positioning problems similar to *Steiner tree* problems.

6.2 Relation to Steiner Tree Problems

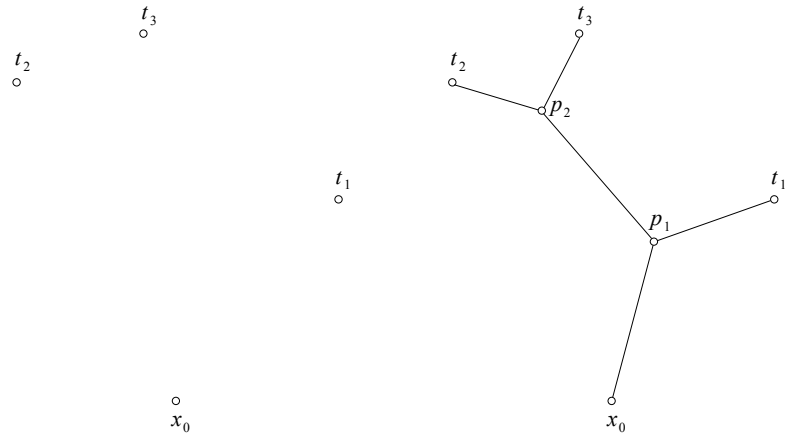
The Steiner tree problem is a well-known optimization problem that exists in both continuous and discrete variants. Steiner tree problems are in general NP-hard and occur in practical applications such as VLSI routing, telecommunication and transportation, and have attracted considerable research efforts. A comprehensive list of applications and algorithms is beyond the scope of this thesis, and we refer to the surveys by Winter [100], Du et al. [38, 37] and Hwang et al. [54].

Common for both the continuous and discrete variants is the term *terminals*, denoted by Z. This is the set of positions that the tree must span, in our case consisting of the base station and all targets. In the continuous problems, this is a set of points, and in the discrete problems it is a set of nodes. In the general Steiner tree problem, the terminals are not required to be leaves in the tree.

6.2.1 Continuous Steiner Trees

The continuous Steiner tree problem consists of connecting a given set of terminals Z by lines of minimum total length so that any two terminals are connected either directly or via other terminals and lines. In the context of the relay problems, $Z = T \cup \{x_0\}$. Any non-terminal point in the Steiner tree, where two or more lines meet, is called a *Steiner point*. Figure 6.1a displays a two-dimensional Steiner tree problem with $Z = \{x_0, t_1, t_2, t_3\}$. A solution with two Steiner points, p_1 and p_2, is displayed in Figure 6.1b. In the continuous Steiner tree problem, there is no need to specify the positions where Steiner points can be located, as all positions can be used.

To be able to solve the general multiple target relay problems, the following requirements all must be handled: i) three-dimensional environments, ii) realistically sized environments, iii) environments with obstacles, iv) cost functions that do not obey the triangle-inequality, since we may sometimes achieve higher quality relay trees through using a larger number of UAVs and longer transmission paths, and v) limited maximum length of lines, corresponding to a limited communication range.



(a) A continuous Steiner tree problem with four terminals.

(b) A solution to the problem in Figure 6.1a, with two Steiner points: p_1 and p_2.

Figure 6.1: A two-dimensional continuous Steiner tree problem.

There are algorithms can handle some of the requirements individually, but not all requirements simultaneously. An example of such an algorithm is the algorithm by Fampa and Anstreicher [42], that can handle three-dimensional environments. Such algorithms can be of interest under very particular circumstances, such as when the reachability functions do not use a range limitation, the cost functions obey the triangle inequality and there are no obstacles in the environment. Similarly, there are algorithms that calculate a Steiner tree in a plane and are able to handle obstacles [101]. Such algorithms are only of interest if the placement of UAVs can be restricted to a plane.

In particular, the requirement that algorithms must be able to handle arbitrary cost functions prohibits the use of current algorithms for continuous Steiner trees. Therefore, we discretize the search space and solve discretized approximations of the multiple target relay problems.

6.2.2 Discrete Steiner Trees

Given a graph $G(N, E)$, the discrete Steiner tree problem consist of finding a minimum cost tree T_G that spans the terminals $Z \subseteq N$, in such a way that all terminals are connected by edges in E. In the continuous Steiner tree problem, Steiner points can be placed anywhere. In the discrete Steiner tree problems, the set of possible positions are the nodes where we can place UAVs in the discretization discussed in Chapter 4. We will later describe

how graphs for use in the discrete Steiner tree problems can be created, as well as different variants of the discrete Steiner tree problem, in both undirected and directed graphs. However, we first introduce some common notation.

Notation. Given a Steiner tree T_G, its nodes and edges are denoted by $N(T_G)$ and $E(T_G)$, respectively. We follow the notation in e.g. [98, 94, 26] and use the term *Steiner nodes* to denote the non-terminal nodes $S(T_G) = N(T_G) \setminus Z$. Each Steiner node corresponds to the use of a UAV. As the number of UAVs is denoted by L (see Section 6.1), $L = |S(T_G)|$ applies and a tree is feasible if $L \leq M$. As for relay chains, $|E(T_G)| = |N(T_G)| - 1$. Thus, as the number of targets is fixed for each problem, and as there must be a single base station, minimizing the number of nodes or edges/hops in a relay tree is equivalent to minimizing the number of UAVs required to realize the tree.

In some nodes in $N(T_G)$, a path "splits" into two or more, which represents the fact that they receive and relay information from several UAVs. As an example, consider x_3 in Figure 1.2 on page 4. Such nodes are called *join nodes* and have two or more outgoing connections. We denote the set of join nodes in T_G by $J(T_G)$. Conceptually, the join nodes correspond to the Steiner points in the continuous problem.

Related Problems. The Steiner tree problem in an undirected graph is similar to the minimum spanning tree problem as both problems require that nodes in a graph are connected, and that the resulting tree has minimum cost. A difference is that $Z = N$ for the minimum spanning tree, as all nodes must be connected in the minimum spanning tree problem. The Steiner tree problem requires that the set of terminals Z are connected and that the cost of the tree is minimized. This makes the general Steiner tree problem more difficult than the minimum spanning tree problem.

While many Steiner tree problems are known to be NP-hard, there are some exceptions. The aforementioned minimum spanning tree is known to be optimally solvable in polynomial time [29]. For $|Z| = 1$, the problem is trivial as only a single node is involved. When $|Z| = 2$, the problem is the shortest path problem, known to be optimally solvable in polynomial time. Thus, both the cheapest path problem and the minimum spanning tree are special cases of the more general Steiner tree problem. The case when $|Z| = 3$ is also possible to solve optimally in polynomial time, which is discussed further in Section 6.5.

Discretization. Creating directed graphs for the multiple target problems can be done using the same method as for a single target, with a few modifications. Replace steps 4 and 7 in the method described in Section 4.1 with the corresponding steps below.

4′. For each target position $t_i \in \{t_1, \ldots, t_l\}$ a new target node τ_i is created. Let $\mathcal{T} = \{\tau_1, \ldots, \tau_l\}$ be the set of all target nodes and let $Z = \mathcal{T} \cup \{n_0\}$ be the set of terminals.

7′. For each $\tau_i \in \mathcal{T}$ corresponding to t_i and for each $x \in U'$ corresponding to $n \in N$ and satisfying $f_{surv}(x, t_i)$, create a directed edge $e = (n, \tau_i)$ of cost $c_{surv}(x, t_i)$ representing the fact that a surveillance UAV at x would be able to surveil the target at t_i.

Undirected graphs can be created in a similar manner as directed graphs. The main difference is in the reachability and cost functions. If such a function is asymmetric, then this is almost certainly due to the fact that the activity that the function is modeling is asymmetric in nature. For example, if a camera is mounted on a UAV's belly and can only see below the UAV, then the surveillance reachability function holds only if the UAV is located above the target. However, it is known in which direction any given surveillance edge will be used, as exactly one of its endpoints is a surveillance target. An undirected surveillance edge can therefore be given a cost corresponding to this specific direction of surveillance.

Most communication functions are symmetric, and many of those that are not can be adjusted to become symmetric. For example, consider the communication cost function based on obstructed volume (Section 3.3.2). The cost is based on the position x only. Let the nodes n and n' correspond to positions x and x' respectively. The cost of the undirected edge between the two nodes can be set to $c_{x,x'} = \frac{V_{ob}(x)+V_{ob}(x')}{2V_{comm}}$. An asymmetric cost function can often be used like this to determine edge costs in an undirected graph.

The Steiner Minimum Tree Problem in Undirected Graphs. The Steiner Minimum Tree (SMT) in an undirected graph is defined as:
Given an undirected graph $G = (N, E)$, an edge cost function $c : E \to R^+$ and a non-empty subset $Z \subseteq N$ of terminals. Find a tree T_G such that there is a path between every pair of terminals and the total cost$(T_G) = \sum_{e \in E(T_G)} c(e)$ is minimized.

As previously mentioned, some restricted variations of the Steiner tree problems such as the cheapest path problem and the minimum spanning tree problem are optimally solvable in polynomial time. As the general

SMT is NP-hard, the time required for calculating optimal solutions is often prohibitively long. For this reason, heuristic algorithms are in many cases used in practice [100].

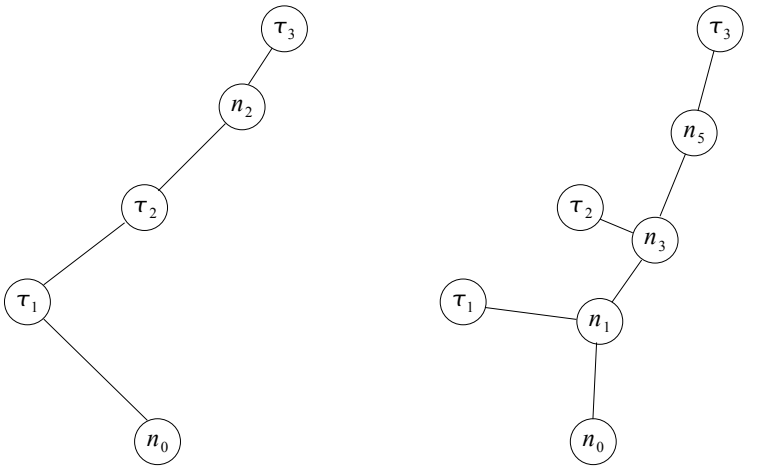
The majority of heuristics for the SMT can be broadly divided into three categories: path heuristics, tree heuristics and vertex heuristics. The path heuristics are based on cheapest path algorithms. The tree heuristics are based on constructing a tree spanning all terminals, which is then improved to decrease the cost. Vertex heuristics attempt to first find good Steiner points, and then calculate a tree spanning the terminals and the Steiner points. For an overview of heuristics for the SMT problem, the reader is referred to Hwang et al. [54].

Heuristics are often chosen on basis of the *approximation ratio* ρ. The approximation ratio guarantees that the cost of the solution calculated by the heuristic is no more than ρ times the cost of the optimal solution. The currently best known approximation ratio for the general Steiner tree problem in an undirected graph is $\rho = 1 + \frac{\ln(3)}{2} \approx 1.55$ [82].

However, in the SMT problem, any terminal can be a join node. In the multiple target relay problems, there is a big difference between the target nodes and the base station node, which may be connected to an arbitrary number of nodes. No target node can be a join node as all targets must be leaves in the relay tree. That is, each target node must be connected to the rest of the tree via a single edge. This is because the targets are arbitrary objects and cannot be used to relay information. The single edge between an inner node and the target node corresponds to a surveillance UAV (the inner node) surveilling a target. Figure 6.2a shows an example which is a valid solution to the SMT but is not a valid relay tree. The target node τ_1 is connected directly to the target node τ_2, which in turn is connected to the node n_2, which is surveilling the target node τ_3. Transferring information from the surveillance UAV at n_2 would require the cooperation of the targets τ_1 and τ_2. The Steiner tree in Figure 6.2b is a valid relay tree as each target node is connected to the rest of the tree by a single edge.

As the SMT problem does not distinguish between terminals and allows any terminal to be connected to multiple nodes, it cannot be used to model the multiple target relay problems. However, the multiple target relay problems can be modeled as several other Steiner tree problems.

Terminal Steiner Trees. The *terminal Steiner tree problem* uses an undirected graph, but requires all terminals to be leaves in the solution. This fits our problem well although it is unnecessarily restrictive as it does not allow several UAVs to communicate directly to the base station. This restriction can be circumvented by adding an extra node, which is connected



(a) In a general Steiner tree, terminals may be connected via several edges.

(b) As each target node is connected to the rest of the tree with a single edge, this Steiner tree is a valid relay tree.

Figure 6.2: Solutions to Steiner tree problems are not necessarily solutions to the multiple target relay problems.

to the base station via an edge with cost zero. Then the extra node is used as a terminal instead of the base station. This allows us to model the multiple target relay problems as a terminal Steiner tree problem.

A generalization of the terminal Steiner tree problem is the *partial terminal Steiner tree problem* (PTSTP), which takes two sets of nodes as input: one set in which all nodes must be leaves and one set of nodes that do not have to be leaves. This fits our problem better as the targets must be leaves while the base station node does not have to be a leaf. Just like the terminal Steiner tree problem, the PTSTP uses an undirected graph. Currently there are only two published algorithms for the PTSTP [51, 52].

The currently best known approximation ratio for the terminal Steiner tree problem is $2\rho - \frac{\rho}{3\rho-2} = 2.52$ (assuming an approximation ratio of $\rho = 1.55$ for the SMT problem). The reason for this approximation ratio is that the algorithm first constructs a Steiner tree and then modifies it to become a terminal Steiner tree, which may increase the cost of the original tree by a certain factor. If edge costs are either 1 or 2, the approximation ratio is lowered to 1.42 [66]. For the partial terminal Steiner tree, the lowest known approximation ratio is $2\rho - \frac{\rho}{3\rho-2} - \epsilon = 2.52 - \epsilon$ (assuming $\rho = 1.55$), where $0 \leq \epsilon \leq \rho - \frac{\rho}{3\rho-2}$ [52].

The algorithms for (partial) Steiner trees require that the triangle inequality applies and that the graph is *complete*. A graph is complete if

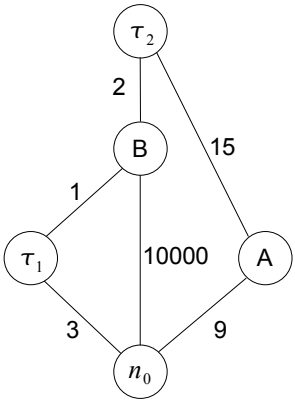
edges between all pairs of nodes exist. Below we discuss the implications of these requirements in detail.

In the basic Steiner tree problem, it is possible to make the triangle inequality apply by changing the cost of any edge that is more expensive than the cheapest path between the nodes. The cost of such an edge is set to the cost of the cheapest path. This causes the triangle inequality to apply, and a solution can be calculated as in the normal case. Then it is checked whether any edge with a changed cost is included in the solution. If so, the edge is replaced by the cheapest path that had the actual cost. However, this method cannot be used in the terminal Steiner tree problems, as shown by Figure 6.3. Here we want to calculate a partial terminal Steiner tree where $\{\tau_1, \tau_2\}$ must be leaves and n_0 does not have to be a leaf. Assume that the algorithm with an approximation ratio of $2.52 - \epsilon$ is used.

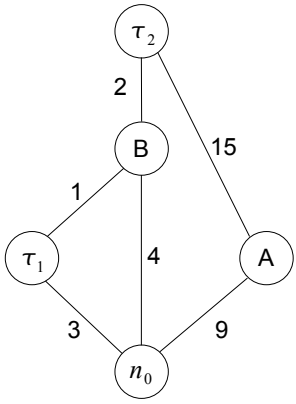
Figure 6.3a shows the original graph and Figure 6.3b shows the graph after the cost of the edge (n_0, B) is set to the cost of the cheapest path between n_0 and B, i.e. 4. Then the algorithm is executed, which is only guaranteed to find a tree with a cost within a factor $2.52 - \epsilon$ of the optimal cost. The optimal tree has cost 9 and the tree found by the algorithm may thus cost up to $9 * (2.52 - \epsilon) = 22.68 - 9\epsilon \leq 22.68$. Assume that the optimal tree is found by the algorithm. However, when edge (n_0, B) is replaced by the cheapest path between the nodes, the tree is no longer a valid partial terminal Steiner tree as the path goes through a terminal. Instead, the real edge (n_0, B) must be used. However, this edge has a cost of 10000, giving a total tree cost of 10005 (Figure 6.3c).

Furthermore, the algorithm can never find the optimal partial terminal Steiner tree, depicted in Figure 6.3d, as it has cost 27 and is outside the guaranteed approximation ratio. This shows that this method for making the triangle inequality apply cannot be used for (partial) terminal Steiner trees and consequently not for calculating solutions to the multiple target relay problems.

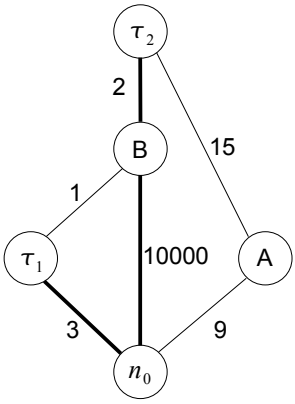
Another option is to only consider instances of the terminal Steiner tree problems where, for each terminal, there is a non-terminal with exactly the same set of neighbors [36]. In such cases, it is possible to transform a terminal Steiner tree problem in which the triangle inequality does not hold into a problem where it does hold. However, this restriction cannot be used in general in the multiple target relay problems. Suppose that when each target node τ_i is created, an extra step is performed, with the intention of placing a node s_i so that it gets the same set of neighbors as τ_i. There must then be some distance between τ_i and s_i, as it must be possible to locate a UAV at s_i without colliding with the target or any other object in the environment. Also, all nodes that could *surveil* the target τ_i must be able



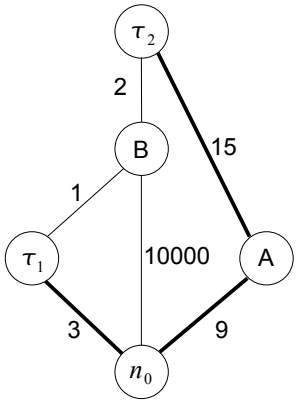
(a) Initial graph



(b) The cost of the edge (n_0, B) is set to the cost of the cheapest path between the nodes.



(c) Solution found when calculating a partial terminal Steiner tree.



(d) The optimal valid terminal Steiner tree cannot be found by the algorithm.

Figure 6.3: Setting the cost of any edge that is more expensive than the cheapest path between the nodes to the cost of the cheapest path, causes the triangle inequality to apply. However, this method cannot be used in the terminal Steiner tree problems.

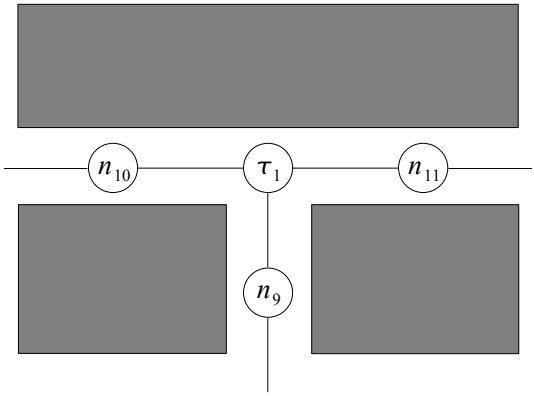


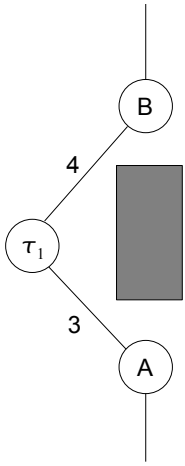
Figure 6.4: In some cases it is impossible to place another node so that it gets the same set of neighbors as a target node. This is especially evident in urban environments.

to *communicate with* s_i. As the communication and surveillance reachability functions may be completely different, this may prevent s_i from getting the desired set of neighbors, especially if we consider the terrain around τ_i. The terrain is most likely to cause problems in urban environments, as exemplified by Figure 6.4. Regardless of where s_1 is placed, it will not get the same set of neighbors as τ_i, when common requirements such as free line-of-sight and limited range are used in the reachability functions.

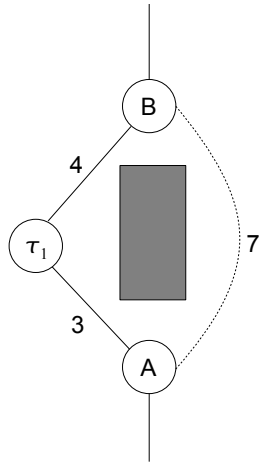
Even if we assume that a cost function obeying the triangle inequality is used, the requirement of a complete graph is difficult to fulfill. The graphs that are used in the relay problems are the results of discretizations of real environments. As such, it is very unlikely that the graphs are complete as there almost certainly are obstacles preventing this. Even if there are no obstacles, the reachability functions generally do not allow the creation of a complete graph due to limited maximum range and similar limitations.

Instead, an incomplete graph must be made complete. One method is to let any missing edge be replaced by the cheapest path between the nodes [36]. This is similar to the above example of making the triangle inequality apply, but the difference is that in that example, there were edges with too high cost, while here there are edges missing.

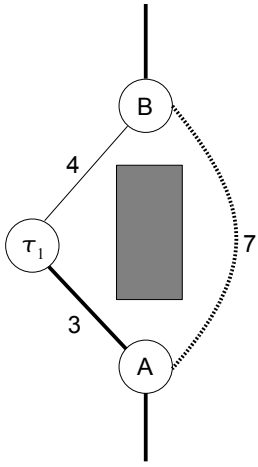
The idea is to first replace the missing edges to make the graph complete, and then execute an algorithm that requires a complete graph. After this is done, any added edges that are used in the solution are replaced by the cheapest path. However, we will show that this method cannot be used in the relay problems as this may cause a path that originally did not pass through a target node to do exactly that.



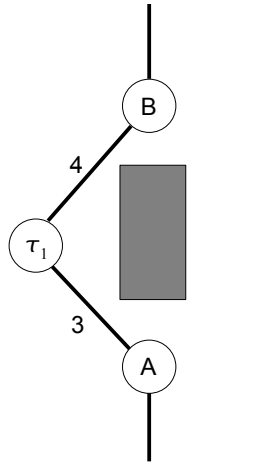
(a) Initial graph



(b) A "pseudo edge" between nodes A and B is added as part of making the graph complete.



(c) The heavy edges are included in a Steiner tree.



(d) The "pseudo edge" is replaced by the real edges, making the tree illegal.

Figure 6.5: In the relay problems, the graph cannot be made complete by adding extra edges consisting of the cheapest paths as this may cause the tree to become invalid.

A part of a graph is displayed in Figure 6.5a: the terminal τ_1 is the only path between the nodes A and B, which in turn are connected to other nodes that are not displayed. As part of making the graph complete, a new, "pseudo edge" between A and B is inserted (Figure 6.5b). The new edge consists of the cheapest path between A and B, in this case $A \rightarrow \tau_1 \rightarrow B$ and the edge cost is the cost of that path. After the graph completion process is performed, a (partial) terminal Steiner tree is calculated. In the tree, both the new edge (A, B) and the edge (A, τ_1) are included (Figure 6.5c). When the "pseudo edge" is replaced by the real edges, τ_1 is no longer a leaf and the resulting tree is not a valid (partial) terminal Steiner tree (Figure 6.5d). Therefore, for the problems of interest here, this method to make the graph complete cannot be used.

The algorithms for the terminal Steiner tree problem and the partial terminal Steiner tree problem require that the triangle inequality holds and that the graph is complete. Unless both these requirements hold, no constant approximation ratio for the terminal Steiner tree problem can be given unless $NP = DTIME(|Z|^{O(\log\log|Z|)})$ [36]. For the PTSTP, two open research questions are whether there exists an approximation algorithm if the triangle inequality does not hold, and if there exists a constant approximation ratio [52].

Neither a complete graph nor a cost function that obeys the triangle inequality can be guaranteed in our problems. Therefore, the algorithms lose their greatest advantages, namely that a solution is guaranteed and that the cost of the solution is guaranteed to be within a certain factor from the cost of the optimum solution. However, it is possible to use algorithms for the *directed Steiner tree* to solve our relay problems.

Directed Steiner Trees. A directed Steiner tree is a possible solution to the problems discussed above, because the algorithms do not require complete graphs or that the cost function obeys the triangle inequality. As a directed Steiner tree uses a directed graph, we can make sure that all targets are leaves by choosing to make all target nodes without edges. The directed Steiner tree problem requires a root node and a set of terminals as input. Naturally, in our case the root node corresponds to the base station node n_0 and the set of terminals is the target nodes $\mathcal{T}$.

The directed Steiner tree problem provides a flexible way to rewrite the relay problems discussed here, as well as other Steiner tree problems. However, the increased flexibility comes with a price, as the heuristics for directed Steiner trees have high time complexities, high error bounds and long execution times [22, 84, 104]. The algorithm by Charikar et al. [22] in some cases requires more than a minute to solve problems with $|N| = 100$ and

$|E| = 400$. To improve this long execution time, Hsieh et al. [50] presented an algorithm that had shorter execution time and yielded trees with similar costs, when tested. However, the algorithm requires $O(|T||N|^2 + h|T||N|)$ memory space, where $h \geq 1$ is the height of the tree. We are interested in solving relay problems in discretizations of real environments, where the number of nodes may be in the tens of thousands and the number of edges in the tens of millions. As the memory requirements for such problems would be prohibitive, such algorithms cannot be used for solving the multiple target relay problems.

Summary. Finding a relay tree is a kind of Steiner tree problem as the total cost of the tree must be minimized. However, the Steiner minimal tree problem does not distinguish between the target nodes and the other nodes in the tree. This allows a target node to be connected to the rest of the tree with several edges, which in the context of the relay problems correspond to using a target to relay information. As this is generally not possible, other ways to model the multiple target relay problems have been investigated.

As existing algorithms for the terminal Steiner trees problems require a complete graph and that the triangle inequality applies, they cannot be used to calculate relay trees. The common ways to make the graph complete cannot be used in the relay problems as they can create invalid relay trees, in which a target node is connected to the rest of the tree with several edges.

Algorithms for solving the directed Steiner tree problem do not have the same requirements regarding a complete graph and the triangle inequality as they use a directed graph. However, the algorithms for solving such problems have either long execution times or high memory requirements.

Considering this, we investigated whether heuristics for the SMT could be modified to calculate relay trees. It turns out that we could adapt the cheapest path heuristic for this purpose.

6.3 Adapting the Cheapest Path Heuristic

The cheapest path heuristic for calculating Steiner trees has been used in both undirected and directed graphs [92, 96]. The heuristic starts with a single node and repeatedly executes a cheapest path algorithm to find paths to the unconnected terminals. During execution of the heuristic, a Steiner tree is incrementally built, where each iteration adds a path from the nodes in the current tree to an unconnected terminal.

The algorithm in its original form is not suitable for calculating relay trees, as it allows target nodes to be connected to several other nodes. However, the algorithm can be modified to fulfill this requirement. The algo-

rithm needs to be changed so that it is guaranteed that each target node is connected to the rest of the tree using a single edge.

To calculate the cheapest paths efficiently, we use a cheapest path algorithm that can handle sets of start nodes and goal nodes, such as Dijkstra's algorithm. In the context of relay problems, the start nodes are the nodes in the tree $N(T_G)$ and the set of goal nodes is the set of target nodes to which no path yet has been found, $\mathcal{T} \setminus N(T_G)$.

Figure 6.6 shows the pseudocode of our modified cheapest path heuristic for calculating relay trees. The relay tree is initialized with the base station node n_0 and an empty set of edges (line 1). The predecessor of n_0 is set to nil. $|\mathcal{T}|$ iterations are performed, as this is the number of targets, and each iteration connects a single target node (line 3). Line 4 clears the priority queue Q. The cost $g(n)$ is initialized for all nodes not in the relay tree (lines 5–6). The set of start nodes is initialized in lines 7–9. For each node, the cost is set to zero and the node is inserted into the priority queue.

Each iteration continues until a path to an unconnected terminal is found (line 16) or the priority queue is empty (line 11). The latter means that the complete graph has been searched and no unconnected terminal has been found. Thus, not all targets can be connected to the tree, and the algorithm returns with failure (line 11). If Q is not empty, the least cost node n is extracted (line 12).

It is checked whether this node is one of the unconnected target nodes (line 13). If so, this means that a cheapest path p to an unconnected target node has been found. The path is retrieved by the function Retrieve-Path(n), and is added to the relay tree, thereby connecting a previously unconnected target node to the relay tree (lines 14–15). This ends one iteration, and the algorithm continues with the next unconnected target node (line 15).

If the node was not an unconnected target node, it is checked whether n is a target node (line 17). This check is necessary because n may be a target node that is already included in T_G. If n is not a target node, then each neighbor node n' is considered with the intention of finding a cheaper path to n' (lines 18–22). If a cheaper path is found, the cost and the predecessor of node n' are updated and n' is inserted into Q. If n' already was in Q, its position is updated due to the decreased cost. The function Extract-tree extracts the complete relay tree so that it can be returned to the user (line 23).

The cheapest path heuristic as described here expands a single tree until all targets have been connected. There are other slightly different heuristics based on repeated execution of a cheapest path algorithm that share the same name and error bound [54].

```
1  T_G ← ⟨n_0, ∅⟩
2  p(n_0) = nil
3  for k= 1, ..., |T| do                    // Connect one more target
4      Q ← ∅                                // Clear priority queue
5      for each n ∈ N \ N(T_G) do
6          g(n) ← ∞
7      for each n ∈ N(T_G) do               // Initialize start nodes
8          g(n) = 0
9          Insert(Q, n)
10     loop
11         if Q = ∅ then return failure     // Not all target nodes are reachable
12         n ← Extract-Min(Q)
13         if n ∈ T \ N(T_G) then           // Found unconnected target node
14             p ← Extract-Path(n)
15             T_G ← T_G ∪ p                // Add path to tree
16             exit loop                    // Continue with next target
17         if n ∉ T then
18             for each n' ∈ n_+ do         // Treat neighbor nodes
19                 if g(n') > g(n) + c_{n,n'} then
20                     g(n') ← g(n) + c_{n,n'}
21                     p(n') ← g(n)
22                     Insert(Q, n')
23 Extract-tree(T)                          // Extract the tree
```

Figure 6.6: Algorithm 4 – Modified cheapest path heuristic for calculating a relay tree.

6.3.1 Theoretical Properties

Algorithm 4 is based on the cheapest path heuristic which is sound and complete for the general Steiner tree problem in both undirected and directed graphs [92]. The pseudocode in Figure 6.6 is an efficient implementation of the cheapest path heuristic with the change that no outgoing connections are created from target nodes (line 17). This corresponds exactly to the requirement in the multiple target relay problems that each target must be a leaf in the tree.

Soundness. The heuristic is sound as each target is connected with a single edge to the rest of the tree. The single connection is assured in line 17 which only considers the outgoing edges of non-target nodes. Furthermore, with the exception of the root node n_0, each node in the tree only has one predecessor, and thus a tree structure is created. When a cheaper path to

a node has been found (line 19), the predecessor node is updated (line 21). Thus, after execution of the algorithm in Figure 6.6, a tree has been created which connects all terminals and each target node is connected to the rest of the tree with a single edge.

Completeness. Consider a problem with target nodes $\tau, \tau' \in \mathcal{T}$. If there is a path from n_0 to τ that does not require going through τ', then it does not matter that the outgoing edges of τ' are not considered, a valid path will be found anyway. If this applies for all $\tau \in \mathcal{T}$, then a valid relay tree will be found, as a path to each such target is found. If the path to τ requires going through τ', then the problem cannot be solved as τ' must be an interior node in the tree. No such tree is a valid relay tree. In that case, the algorithm is not required to return a solution.

Time Complexity. The algorithm has a time complexity of $O(|\mathcal{T}|(|E| + |N| \log |N|))$ as $|\mathcal{T}|$ executions of a cheapest path algorithm are performed, each one with a time complexity of $O(|E| + |N| \log |N|)$.

Approximation Ratio. The approximation ratio is $|\mathcal{T}|$ in both directed and undirected graphs [96]. Despite this approximation ratio, the cheapest path heuristic has been known to be competitive with more advanced methods in directed graphs [50, 96].

6.3.2 Extensions

The basic algorithm as shown in Figure 6.6 can easily be extended to find relay trees for a variety of restrictions on how UAVs may be used. These extensions require the addition of a Boolean function Acceptable-Connection(n, n') that holds if certain conditions are fulfilled, discussed further below. The function is added to the if-statement in line 19, which will then be:
if $g(n') > g(n) + c_{n,n'}$ **and** Acceptable-Connection(n, n') **then**.

Next we will give some examples of the specific restrictions and how these are implemented using the function Acceptable-Connection.

Relay UAVs Have Limited Capacity. If the relay UAVs can only relay a limited amount of information, then an important question is whether all information streams require the same amount of bandwidth or if it differs. If all information streams require equal amounts of bandwidth, then each relay UAV must be able to handle a certain number of connections. Adding a new information stream requires that all UAVs between the first relay UAV and the base station can handle the additional stream of information.

If information streams instead require different amounts of bandwidth, then the same UAVs must be able to handle the additional bandwidth.

To implement this requirement, the function Acceptable-Connection could check whether $n' \in \mathcal{T} \setminus N(T_G)$, i.e. if n' is an unconnected target. If this is true, then a surveillance UAV may be placed at n if all nodes in the path q between n and n_0 can handle an additional stream of information. This is controlled by retrieving the path q and checking each node in q for available relay capacity. If all nodes have available capacity, then the information can be relayed back to the base station and the target n' can be surveilled from a surveillance UAV at n. Otherwise, a relay chain to n' with the surveillance UAV at some other position must be found.

Surveillance UAVs May Not Relay. If surveillance UAVs have limited energy, then restricting the amount of information that they transmit is one way to extend the time that a target can be surveilled. Naturally, each surveillance UAV must transmit information from its own surveillance, but should not relay any other information.

This restriction is different from the restriction that relay UAVs have limited capacity. Here the surveillance UAVs cannot relay any information at all, while relay UAVs still have unlimited capacity.

If a surveillance UAV cannot be used to relay information, then each surveillance UAV must have one connection to a predecessor (relay UAV or base station) and one or more connections to surveillance targets. As n is in T_G, it has a predecessor and it is sufficient to check if the currently treated node $n' \in n_+$ is a target node. If so, then the edge (n, n') is a surveillance edge and may be added to the tree. If not, the edge corresponds to a communication edge and is not allowed. This requirement can be implemented in the function Acceptable-Connection which holds if $n' \in \mathcal{T}$.

Limited Target Distance. A surveillance UAV can only surveil multiple targets within a limited distance from each other. Therefore, when a surveillance UAV is already surveilling one or more targets and it is checked whether it can surveil an additional target, the distance between the targets currently under surveillance is calculated and compared to the maximum allowed distance. If the distance is below the limit, then the additional target may also be surveilled.

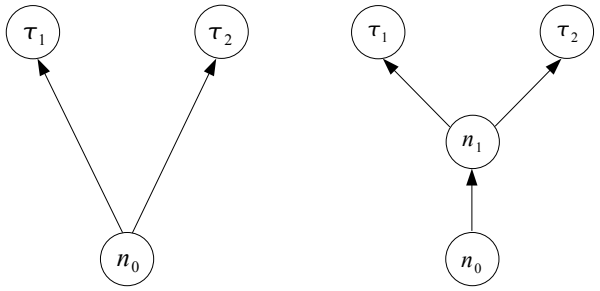
These requirements are checked by the function Acceptable-Connection. The neighbor node n' is an unconnected target if $n' \in \mathcal{T} \setminus N(T_G)$. Retrieve the possibly empty set $E_{surv,n}$ consisting of all edges $(n, \tau_i) \in E(T_G),\ \forall \tau_i \in \mathcal{T}$. Then, all τ_i in $E_{surv,n}$ are target nodes and arbitrary restrictions regarding which targets that can be surveilled can be applied.

6.4 Calculating Pareto-optimal Relay Trees For Two Targets

Trees calculated with the modified cheapest path heuristic often have good quality, but there might still be room for improvement. One possible option is to incrementally improve the tree by optimizing subtrees. For example, we can choose subtrees with a root node and two leaves and perform local optimization on such trees. In this section we present a method that calculates Pareto-optimal relay trees for the case of a base station and two targets (leaves). In a later section, this algorithm will be generalized and used to optimize subtrees in larger relay trees.

In optimal Steiner trees, there can be at most $|Z| - 2$ join nodes, and therefore a relay tree with $|Z| = 3$ has *at most* one join node. Obviously, it must have one join node, as the paths to τ_1 and τ_2 both originate in n_0. The fact that an optimal Steiner tree with three terminals has exactly one join node allows us to solve the **MTR-ParetoLimited** problem optimally. We can first calculate all Pareto-optimal relay chains to each node, and then connect them so that all Pareto-optimal Steiner trees are created. A Steiner tree with a single join node has either a *v-structure* or a *y-structure*. If n_0 is the only common node in the paths to τ_1 and τ_2, then the tree has a v-structure (Figure 6.7a). Otherwise, the paths have at least one more common node, and the tree has a y-structure (Figure 6.7b).

For two targets, both the **MTR-MinCostMinLength** and the **MTR-MinLengthMinCost** problems can be solved optimally with a time complexity of $O(|E| \log |E|) \subseteq O(|N|^2 \log |N|^2)$ through the algorithm by Chen [23]. However, for the local optimizations that will be described in Section 6.5, we are more interested in solving the **MTR-ParetoLimited** prob-



(a) A tree with a v-structure. (b) A tree with a y-structure.

Figure 6.7: Structures of trees with one join node.

1 Execute Algorithm 2 starting in n_0.
2 Execute Algorithm 2 starting in τ_1.
3 Execute Algorithm 2 starting in τ_2.
4 **for** each $n \in N \setminus \mathcal{T}$ **do**
5 Construct relay tree(s) with join node n from the reachability records in n.
6 Return the set of Pareto-optimal relay trees.

Figure 6.8: Algorithm 5 – Solving the two targets relay problems through multiple executions of Algorithm 2.

lem as this allows us to choose other trees than the cheapest or shortest. We solve the **MTR-ParetoLimited** problem through executing Algorithm 2 three times, and then calculating all possible valid Steiner trees. Between each pair of nodes, there are at most $|N|$ different Pareto-optimal paths. As there are three terminals in the tree, this yields at most $|N|^3$ Pareto-optimal trees in each node. Determining this for all $|N|$ nodes yields a time complexity of $O(|N|^4)$. Executing Algorithm 2 three times is in $O(|N|^3)$ and selecting the best relay tree is in $O(|N|)$. Thus the total time complexity is $O(|N|^4)$. However, in practice there are far fewer than $|N|$ Pareto-optimal paths to each node, and the time complexity is a severe overestimate.

As the Steiner tree must connect all three nodes, only nodes that are reachable from both n_0, τ_1 and τ_2 are of interest as join nodes. If no such node exists, then no tree can connect the base station and the target nodes and the problem cannot be solved.

We refer to this method of calculating relay trees for a base station node and two target nodes as Algorithm 5. Figure 6.8 displays the pseudocode of the algorithm. For each execution of Algorithm 2, a set of reachability records is created in each reachable node. Thus, after three executions of Algorithm 2 (lines 1–3), three sets of reachability records exist in the nodes that are reachable from both n_0, τ_1 and τ_2. Then, an iteration through all reachable nodes, except τ_1 and τ_2, is performed. In each node, the set of relay trees is calculated by repeatedly selecting and combining a reachability record from each set (lines 4–5), described in more detail in Section 6.4.1.

Special care must be taken if Algorithm 5 is used in a directed graph. The executions of Algorithm 2 in lines 2–3 will use the nodes' incoming edges instead of outgoing. This is because in the final tree, the edges will be used to provide paths *to* the target nodes, not from them.

k	g_k	p_k
1	50	n_0
2	32	n_1
4	21	n_2

k	g_k	p_k
1	13	τ_1

k	g_k	p_k
1	19	τ_2
2	15	n_{23}

Table 6.1: Reachability records for executions starting in nodes n_0, τ_1 and τ_2, respectively, for some node n.

6.4.1 Determining the Set of Pareto-optimal Relay Trees

We will now show how the reachability records are combined to create relay trees.

Assume that after Algorithm 2 has been executed three times, once starting in the base station node and once in each target node, the set of reachability records displayed in Table 6.1 exists for some node $n \in N \setminus \mathcal{T}$. For example, there are three paths from n_0 to n.

By repeatedly selecting and combining Pareto-optimal paths from the different sets, information about $3 \times 1 \times 2 = 6$ trees is created. All six relay trees have node n as join node, and we calculate the total costs and lengths by adding the costs and lengths of the individual chains.

By looking at the resulting information about the trees (Table 6.2), it is evident that some trees are inferior. For example, there is no reason to use the tree with total path length 5 and total cost 60 as it is more expensive than the tree with the same length and cost 53. The inferior trees can be removed by evaluating the trees in a manner similar as for the Pareto-optimal relay chains, see Section 3.2.

The above process finds all Pareto-optimal trees with join node n. The process is then repeated for all nodes $N \setminus \mathcal{T}$, as no target node can be a join node. The complete set of Pareto-optimal trees, possibly with different join nodes, can be found as follows. Start with an empty set of Pareto-optimal trees. Then for each non-target node n, consider the trees that could be generated with n as a join node according to the reachability records in that node. For each such tree, check whether there is already a better (in the Pareto-optimal sense) tree in the current set of trees. If not, the tree is added to the set and any trees dominated by the new tree are removed. The set of Pareto-optimal trees together form the solution to the **MTR-ParetoLimited** problem. Table 6.3 shows such a set.

Path length					
to n_0	to τ_1	to τ_2	total	Cost	Pareto-optimal
1	1	1	3	82	Yes
1	1	2	4	78	No
2	1	1	4	64	Yes
2	1	2	5	60	No
4	1	1	5	53	Yes
4	1	2	7	49	Yes

Table 6.2: Information about the different relay trees joining in node n.

Total path length	Total cost	Join node
3	82	n
4	64	n
5	51	n''
7	49	n
8	47	n''

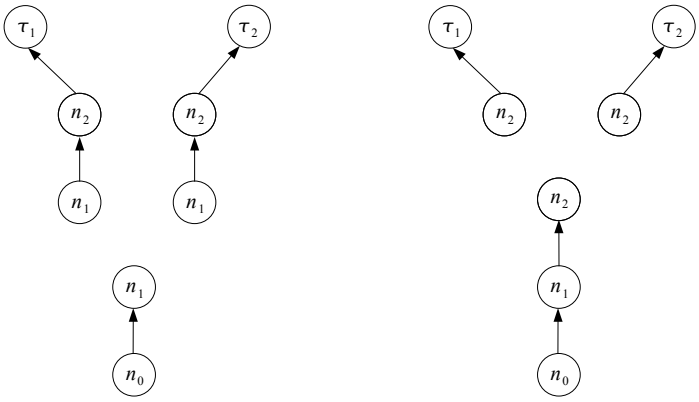
Table 6.3: The set of Pareto-optimal relay trees.

6.4.2 Duplicate Edges in the Relay Tree

In rare cases, it may happen that two or more paths in a relay tree coincide in more nodes than just the join node, i.e. there are also common edges. Figure 6.9a shows an example where three paths have been calculated to the join node n_1. As our calculations simply sum the costs and lengths of all paths involved, the cost of the edge (n_1, n_2) will be counted twice. This tree will then be considered at least as expensive, and use one hop more, than the tree in Figure 6.9b. The latter tree has n_2 as join node.

It might seem like all trees need to be checked for duplicate edges and subtract the extra cost and hop of any such edges. However, regardless of whether the graph is directed or undirected, nothing needs to be done. When iterating through all non-target nodes to find the join node for the tree, a tree that includes the edge (n_1, n_2) and its cost only once will be found. In this example, the tree has join node n_2.

Assume that the tree in Figure 6.9a has been found. In a directed graph, the edge (n_1, n_2) is guaranteed to exist as it has been used in the paths for τ_1 and τ_2. The cost of the edge and the hop required to use the edge has been included in the tree twice. Naturally, using the edge once cannot be more expensive than using it twice. In addition, using the edge once also gives



(a) The paths that are combined to form a relay tree are displayed separately for clarity. The join node is n_1.

(b) The tree with n_2 as join node will be preferred as it will be cheaper.

Figure 6.9: Duplicate edges can occur when reachability records are combined to form a tree, but there always exists a cheaper relay tree without the duplicates.

a tree with one less hop. Therefore, a cheaper tree with one less hop must exist (Figure 6.9b). Such a tree will be found when iterating through all non-target nodes to find the join node of the tree. The same reasoning applies in an undirected graph, where $c_{n_1,n_2} = c_{n_2,n_1}$ obviously holds. Therefore, it must be the case that $c_{n_1,n_2} \leq 2c_{n_2,n_1}$, with equality for cost zero. In other words, it will never be more expensive to use the edge once than to use it twice. Similarly, a cheaper and shorter tree must exist and will be found when determining the join node of the tree.

6.5 Improving Relay Trees

Algorithm 5 creates Pareto-optimal relay trees with one join node. This can also be used to optimize larger trees with one join node and is used in a heuristic improvement algorithm that can be applied to optimizing relay trees once an initial relay tree has been calculated [77].

The algorithm works by performing a series of local optimizations of the existing tree. In each such optimization, a subtree with one join node is chosen for optimization. Then a set of candidate subtrees is calculated to replace it. Each such candidate subtree also has one join node. From the set of candidates, the best subtree according to some *optimization criterion* (described further below), is chosen to replace the existing subtree. Then, the new subtree is compared to the existing subtree and a replacement

is performed only if the new subtree is better than the existing subtree with respect to the optimization criterion. As the number of hops and the cost of the complete tree is the sum of the costs/hops of all subtrees, any improvement of a subtree improves the complete tree. When a replacement has been performed, the new and improved tree can be displayed to the user.

The process of continually optimizing a relay tree can be performed until no better tree is found, or until the available time runs out, if there is a time limit.

Optimization Criterion. Subtrees are optimized with respect to a certain optimization criterion. Examples of such criteria are minimizing the number of hops in the tree or finding the least cost tree, regardless of the number of hops, or the cheapest tree with a limit on the number of hops in the tree.

The algorithm permits the use of different optimization criteria, and the criteria can also be changed during the course of optimizing the relay tree. For example, assume that we want the least cost feasible tree. If the original tree is infeasible, then the optimization criterion of minimizing the number of hops in the tree is used until a feasible tree is found. At that time, the optimization criterion is changed to finding the least cost tree with a limit on the number of UAVs, to assure that the tree remains feasible. If the initial tree was feasible, the optimization criterion would be to find a least cost tree with the restriction that the tree must remain feasible.

The above optimization criteria correspond to some extent to the three first multiple target relay problems, and are typically used when trying to calculate approximate solutions to these problems. This algorithm is similar to an anytime algorithm [103], in the sense that it continually improves a solution as time goes on.

Notation. Let a subtree of T_G be denoted by T_s, with $r(T_s)$ denoting the root node and $L(T_s)$ denoting the set of leaves of the subtree. A subtree that is a candidate for replacing T_s is denoted by $T_s^{'}$. Naturally, each candidate subtree $T_s^{'}$ has the same root node and the same set of leaves as the subtree it is intended to replace: $r(T_s) = r(T_s^{'})$ and $L(T_s) = L(T_s^{'})$. The set of join nodes in each subtree T_s consists of exactly one join node and we use the variable $J(T_s)$ interchangeably for the set of join nodes in T_s and the join node itself.

6.5.1 Reduced Trees

Only a few nodes are necessary to characterize the structure of T_G. The set of such nodes is referred to as *key nodes*. These nodes are the terminals together with the join nodes.

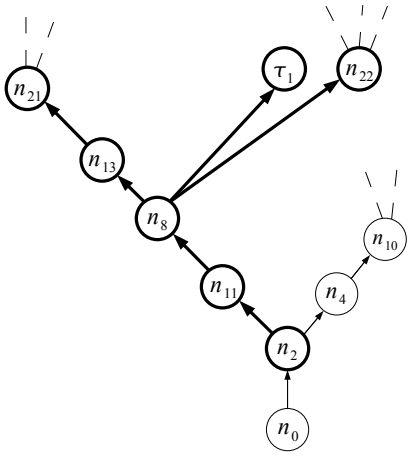
To allow for quickly determining the subtrees for optimization, a *reduced tree* is created. The reduced tree is created by finding all paths in the tree that start in one key node and end in another key node. Each such path is then replaced by a single edge. Thus, the reduced tree consists of only key nodes and maintains the same topology as the relay tree.

Figure 6.10a displays a part of a relay tree, and the corresponding reduced tree is displayed in Figure 6.10b. It is clear that the reduced tree retains the same topology as the original tree. An optimization of the subtree with $J(T_s) = n_8, r(T_s) = n_2$ and $L(T_s) = \{n_{21}, \tau_1, n_{22}\}$, marked by the heavy lines, is performed. A better subtree is found and the previous subtree is replaced. The tree after replacement is displayed in Figure 6.10c, and the corresponding reduced tree is displayed in Figure 6.10d. The new subtree is marked by heavy dashed lines in both figures.

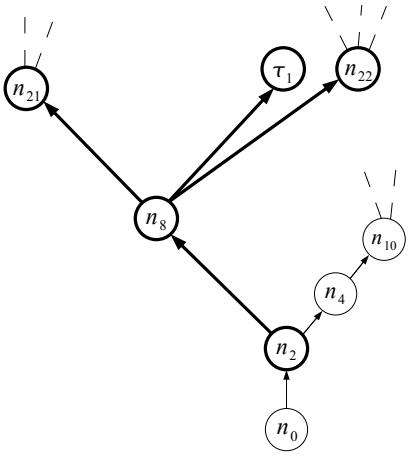
6.5.2 Choosing Subtrees for Optimization

In each iteration, the algorithm chooses a join node $J(T_s)$ whose subtree T_s will be optimized. The reduced tree is used to determine the root node $r(T_s)$ and set of leaves $L(T_s)$ of T_s. The leaves are all the immediate successors of $J(T_s)$ in the reduced tree. The reason why all leaves are chosen is discussed further below. In most cases, $r(T_s)$ is the immediate predecessor of $J(T_s)$. Such a subtree has a y-structure (Figure 6.7b), and subtrees with such structure are chosen whenever possible.

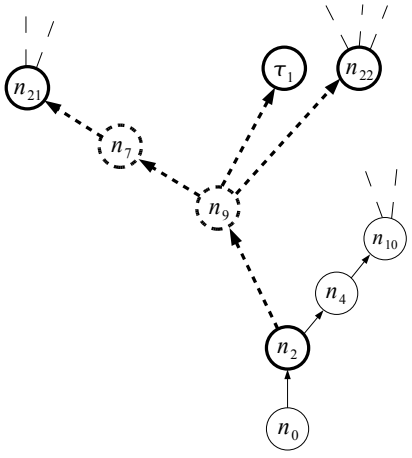
If $J(T_s) = n_0$, there are two possibilities (Figure 6.11). If no immediate successor of n_0 in the reduced tree is a target, then no optimization with n_0 as join node needs to be performed. For an example of this, consider the tree in Figure 6.11a. In that tree, there are three join nodes, n_0, n_2 and n_3. However, only two subtree optimizations need to be performed. These are the trees with n_2 and n_3 as join nodes, respectively. Together these optimizations are sufficient to optimize all parts of the tree. If any of n_0's immediate successors in the reduced tree is a target, then an optimization of a subtree with n_0 as join node needs to be performed. Consider Figure 6.11b for an example of this. In addition to the two optimizations with n_2 and n_3 as join nodes, a third optimization with $J(T_s) = r(T_s) = n_0$ needs to be performed. The leaves in that subtree would be $\{n_2, n_3, \tau_5\}$ and the subtree has a v-structure (Figure 6.7a). The third optimization is required so that all subtrees have been subject to optimization.



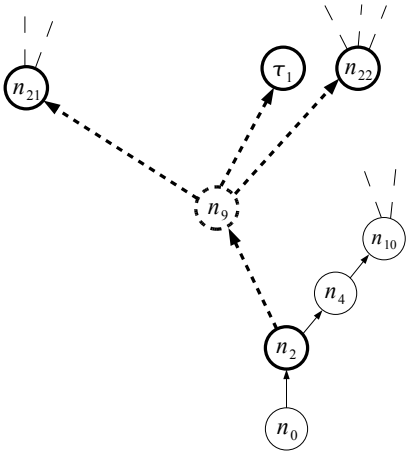
(a) A part of a relay tree.



(b) The reduced tree corresponding to the tree in Figure 6.10a.

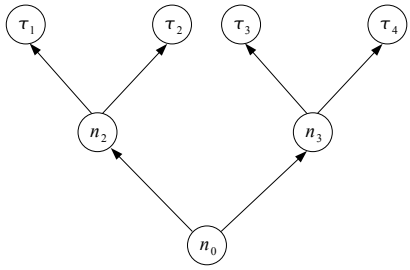


(c) The new subtree, marked by heavy dashed lines, replaces the old subtree.

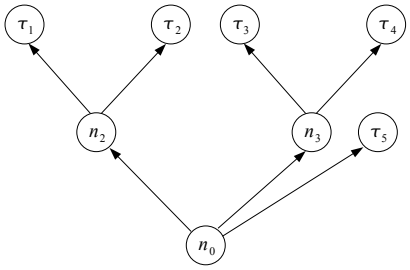


(d) The reduced tree corresponding to the tree in Figure 6.10c.

Figure 6.10: Part of a relay tree and the corresponding reduced tree, before and after optimization. The subtree marked by heavy lines is replaced by the tree marked by heavy dashed lines.



(a) In this tree, only two optimizations are needed.

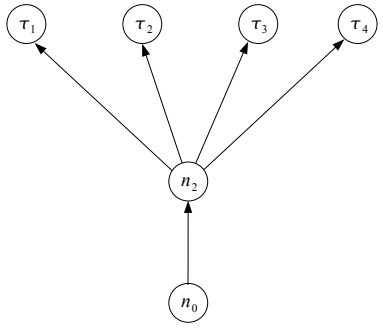


(b) This tree requires three subtree optimizations to allow that all parts of the tree have been subject to optimization.

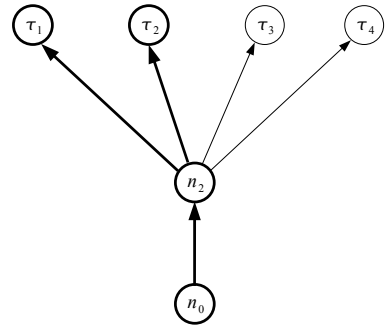
Figure 6.11: Two different cases can occur when n_0 is a join node.

The fact that each join node is chosen in some iteration, together with the structure of the chosen subtrees permits the replacement of all non-terminal nodes.

The reason for including all immediate successors of the join node is exemplified in Figure 6.12. The reduced tree has one join node, n_2 (Figure 6.12a). If a subset of a join node's leaves were chosen for optimization, then several optimizations with the same join node would need to be performed, to allow that all paths in the tree are optimized. Consider Figure 6.12b, where the subtree with $r(T_s) = n_0, J(T_s) = n_2$ and $L(T_s) = \{\tau_1, \tau_2\}$ is to be optimized. This subtree is marked by the heavy



(a) Original reduced tree.



(b) If choosing a subtree with $L(T_s) = \{\tau_1, \tau_2\}$, the join node n_2 must remain as it is connected to the nodes τ_3 and τ_4.

Figure 6.12: Including the complete set of a join node's leaves in a subtree optimization potentially allows a better result when the subtree is optimized.

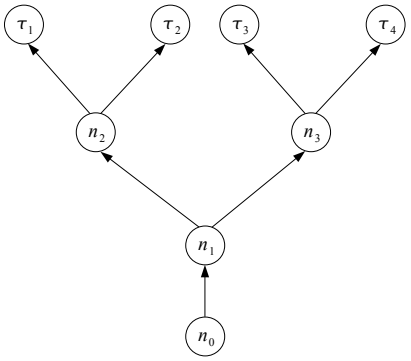
lines. When performing this optimization, the join node n_2 cannot be replaced as it is connected to the leaves τ_3 and τ_4. In that case, only the paths to and from n_2 can be optimized. If instead the complete set of leaves is chosen when optimizing the subtree, then the join node can also be replaced. This potentially permits better trees and is the reason why we include all immediate successors of the join node in subtree optimizations.

It is possible that the complete tree would be chosen for optimization in one iteration. However, for that to happen, the tree would need to have a very specific structure, i.e. all targets would be connected directly to n_0 without any intermediate join nodes.

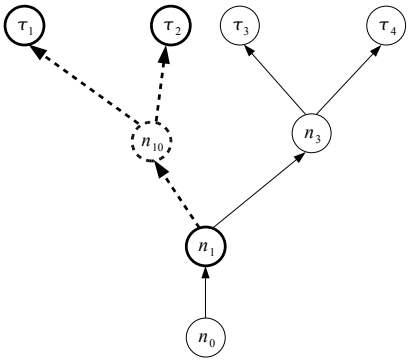
We will now give an example of how join nodes are chosen and how this affects the tree over several iterations. Consider the reduced tree in Figure 6.13a. For example, let the subtree with $r(T_s) = n_1, J(T_s) = n_2$ and $L(T_s) = \{\tau_1, \tau_2\}$ be chosen in the first iteration. During optimization of that subtree, only the root and leaf nodes are fixed: the join node and all other nodes in the subtree can be replaced. Assume that during this optimization, a subtree with the join node n_{10} replaces the old subtree (Figure 6.13b). In the second iteration, an optimization of the subtree with $r(T_s) = n_1, J(T_s) = n_3$ and $L(T_s) = \{\tau_3, \tau_4\}$ is performed. This finds a new subtree with join node n_{21} (Figure 6.13c). The final optimization of the tree involves the subtree with $r(T_s) = n_0, J(T_s) = n_1$ and $L(T_s) = \{n_{10}, n_{21}\}$. This subtree's leaves are the resulting middle nodes from the previous optimizations. Thus, all paths in the tree have been subject to optimization at least once (Figure 6.13d).

A successful optimization of one subtree can allow for further optimization of other subtrees. The condition is that at least one node in the optimized subtree is a non-leaf node in another subtree. The reason for this is that the root node and leaves are fixed in each optimization, and cannot be replaced. As an example, consider Figure 6.13d where a new subtree has replaced the old subtree. Here the join node has changed from n_1 to n_8. As subtrees are chosen to be partially overlapping, this opens up the possibility of further optimizing subtrees involving the node n_8. Here there are two such subtrees, one subtree with $r(T_s) = n_8, J(T_G) = n_{10}$ and $L(T_s) = \{\tau_1, \tau_2\}$ and another subtree with $r(T_s) = n_8, J(T_G) = n_{21}$ and $L(T_s) = \{\tau_1, \tau_2\}$. If any of these trees are optimized and the join node is replaced, it opens up the possibility of optimizing the tree with $r(T_G) = n_0$ again. This shows an example of the fact that the number of subtree optimizations for a tree is not fixed, but depends on how the optimization of different subtrees affects other subtrees.

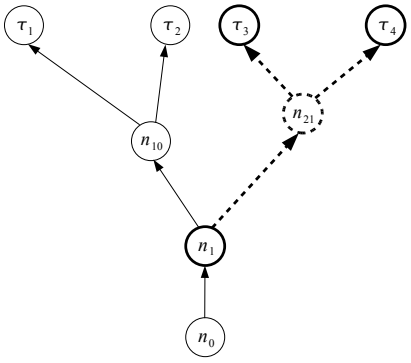
The order in which subtrees are optimized can be chosen in many different ways, for example starting with the subtrees furthest from the root



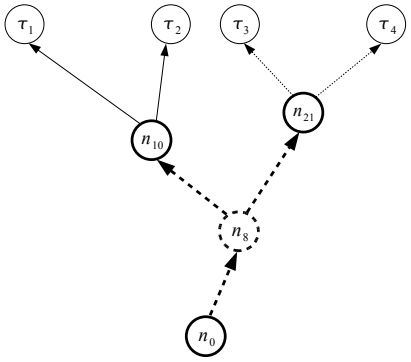
(a) Initial reduced tree.



(b) Optimization of the subtree with $r(T_s) = n_1$ and $L(T_s) = \{\tau_1, \tau_2\}$ leads to a new subtree with join node n_{10}.



(c) A new tree, with join node n_{21} is found for the tree with $r(T_s) = n_1$ and $L(T_s) = \{\tau_3, \tau_4\}$.



(d) The final optimization has the two new join nodes as leaves.

Figure 6.13: By choosing subtrees with y-structures for optimization whenever possible, all paths in the tree are subject to optimization.

node and progressing towards the root node.

6.5.3 Different Tree Structures

All subtrees that are chosen for optimization have one join node, and such trees can have either a y-structure or a v-structure. The y-structure is the most common and occurs in a majority of cases. In such a subtree, the root node is either n_0 or a join node. The leaves are either join nodes or targets, or a combination thereof. A basic example of such a tree is displayed in Figure 6.7b on page 83. The subtree marked by heavy lines in Figure 6.10d displays another example of such a tree. The root node is n_2, the join node is n_8 and the leaves are $\{n_{21}, \tau_1, n_{22}\}$. The root node, the join node and two of the leaves are join nodes in T_G. This example shows a generalized y-structure with more than two leaves.

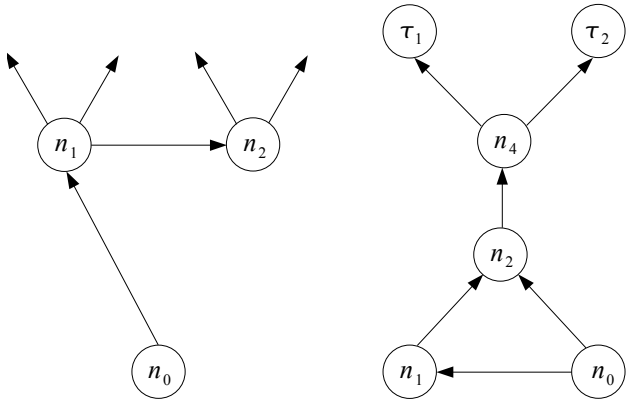
A subtree chosen for optimization can also have a v-structure (Figure 6.7a on page 83). This occurs only when the join node of the subtree coincides with n_0. The leaves of the subtree can be join nodes or targets or a combination.

It is possible that the subtree T_s has one structure and the candidate subtree $T_s^{'}$ that replaces T_s has a different structure. This poses no problem as the root node and the set of leaves is the same for the two trees. Therefore, the new subtree can be inserted into T_G. The candidate $T_s^{'}$ can have either a y-structure, a v-structure, an *l-structure* or an *x-structure*.

The l-structure (Figure 6.14a) occurs when a leaf in T_s becomes the join node in $T_s^{'}$. Obviously, this structure can only occur when the new join node is a non-target node, as the join node must be able to connect the other leaves.

On rare occurrences, when treating a subtree with three or more leaves, it can happen that the paths from the root node to the leaves split and join several times (Figure 6.14b). This is no longer a valid tree as at least one node has multiple predecessors, due to several paths joining in the node. Each node with multiple predecessors is a conflict that must be resolved, so that each node only has one predecessor. The possible exception to this is if the subtree's root node is n_0, which of course has no predecessor.

Solving a conflict involves evaluating each path with respect to the chosen optimization criterion, and removing all but the best path. However, it is not necessary to evaluate the complete paths: it is sufficient to evaluate the paths from the last common node to the conflicting node. The best path is kept and all other paths are removed. As the best path is kept, the quality of the subtree does not deteriorate. After the worse paths have been removed for all conflicts, the result is a valid subtree.



(a) A tree with an l-structure.

(b) An invalid tree with an x-structure.

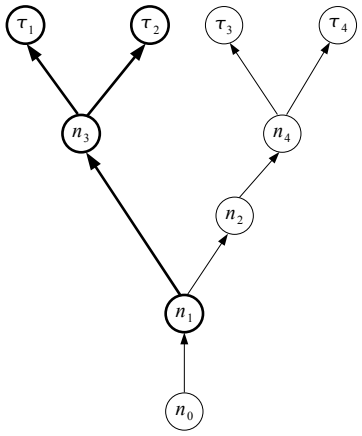
Figure 6.14: The candidate subtree $T_s^{'}$ can have any of the above structures as well as any of the structures in Figure 6.7.

The number of join nodes can change during the course of optimization. If T_s has a y-structure and $T_s^{'}$ has either a v-structure or an l-structure, then a join node has been removed and the number of join nodes decreases by one. If T_s has a v-structure and $T_s^{'}$ has either a y-structure or an l-structure, then a join node has been added and the number of join nodes increases by one. A change in the number of join nodes does not affect the number of subtree optimizations that is performed in the basic algorithm. However, if the algorithm is changed to continue to perform optimizations of subtrees that have been optimized previously, the changing number of join nodes will have an effect on the number of subtree optimizations that are performed. See Section 6.5.5 for a discussion about this change.

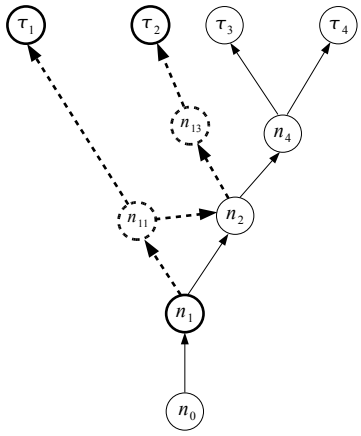
6.5.4 Collisions Between Trees

When optimizations are performed, it is possible that the new subtree "collides" with other parts of the tree. That is, one or more nodes in the new subtree are also part of the tree that is not currently being optimized. Figure 6.15 shows an example of a tree collision and the different situations that can occur. The subtree marked by heavy lines in Figure 6.15a is to be replaced by the subtree marked by heavy dashed lines in Figure 6.15b. However, the new subtree has node n_2 in common with the part of the tree that is not currently being optimized.

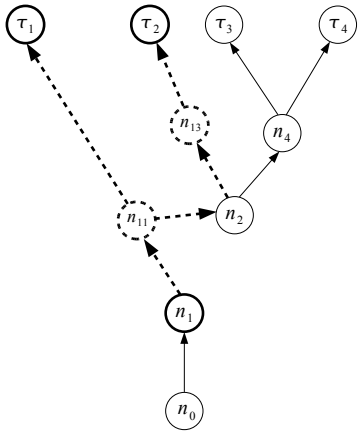
Just like the conflicts within the subtree, the collision causes the relay tree to violate the definition of a tree and must be corrected. There are



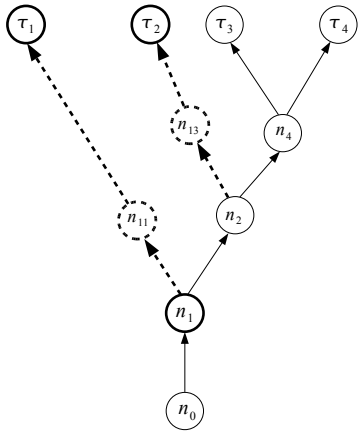
(a) The subtree marked by heavy lines is optimized.



(b) Node n_2 is part of both the old unoptimized tree and the new subtree.



(c) If the new path in to n_2 is better, parts of the old tree are removed.



(d) If the old path to n_2 is better, parts of the new subtree are removed.

Figure 6.15: Collision between the new subtree and the rest of the relay tree.

several different ways to solve this: a simple way is to mark nodes in $N(T_G) \setminus N(T_s)$ as not being allowed in $N(T_s^{'})$. However, this might lead to lower quality trees as some nodes are removed from use.

Another alternative that potentially gives better trees, but requires an additional step, is to allow all nodes, and check whether the new subtree $T_s^{'}$ contains one or more nodes that are part of the tree that is not being optimized. For each such node, we must determine which path to the node to use.

Consider Figure 6.15b, with the colliding node n_2. Here we must determine whether it is better to use the old or the new path to n_2. To do this, a traversal is performed from n_2 towards n_0, but stopping at the first node in the tree that must remain even if the path to n_2 is removed. Such an unchanging node can be either a join node or n_0 if no join node is encountered. The traversal is performed in both the original tree and in $T_s^{'}$. In $T_s^{'}$, the path is $n_{11} \rightarrow n_2$ and in the original tree, it is $n_1 \rightarrow n_2$. It does not matter if the first unchanging nodes are different. It is never necessary to remove more than the path to such a node, as the join nodes have several successors and must thus remain in the tree even if one successor is removed, and the base station node n_0 is never removed. If the path in $T_s^{'}$ is better according to the optimization criterion, then the path in the old tree is removed, and the path in $T_s^{'}$ remains (Figure 6.15c) and vice versa (Figure 6.15d).

If there are several collisions, they can be handled sequentially, and the updated tree is used when determining the paths to the join nodes, thus the currently best tree is used at all times.

6.5.5 Algorithm Details

To optimize subtrees, Algorithm 5 is generalized. The pseudocode for the new algorithm, Algorithm 6, is displayed in Figure 6.16. The preference relation $T_s^{'} \prec T_s$ holds if $T_s^{'}$ is better than T_s with respect to the optimization criterion.

As described in Section 6.5.2, choosing a subtree for optimization is a matter of selecting a join node and extracting the corresponding subtree. We keep track of the set of join nodes whose subtree has not been optimized. This set of untreated join nodes of T_G is denoted by $J_u(T_G)$. Initially $J_u(T_G) = J(T_G)$.

Systematically optimizing the complete tree requires optimizing all subtrees. This corresponds to $|J_u(T_G)|$ subtree optimizations (line 1). Each subtree optimization begins with choosing a join node $J(T_s)$ (line 2). Once a join node has been chosen, it is removed from $J_u(T_G)$ as it may not be chosen again (line 3).

```
1  for k = 1, ..., |J_u(T_G)| do
2     J(T_s) ← Choose-join-node(J_u(T_G))
3     J_u(T_G) ← J_u(T_G) \ J(T_s)
4     T_s ← Extract-subtree(J(T_s))
5     Execute Algorithm 2 starting in r(T_s)
6     for each n ∈ L(T_s) do
7        Execute Algorithm 2 starting in n
8     for each n ∈ N \ T do
9        Calculate-cost-for-subtrees(n)
10    T_s' ← Choose-best-subtree
11    if T_s' ≺ T_s then
12       T_G ← T_G \ T_s          // Remove old subtree
13       T_G ← T_G ∪ T_s'         // Insert new subtree
14       Yield T_G                // Yield improved tree
```

Figure 6.16: Algorithm 6 – Algorithm for optimizing existing relay trees.

The subtree T_s in which $J(T_s)$ is the join node is extracted from the reduced tree (lines 4). Then Algorithm 2 is executed once starting in the subtree's root node and once in each of the subtree's leaf nodes (lines 5–7). Each execution creates a set of reachability records in each reachable node.

Similar to Algorithm 5, only nodes reachable from both the root node and the subtree's leaves are of interest when the join node is determined. Target nodes are excluded from the calculation of subtrees, as the target nodes must be connected to the tree with a single edge, and join nodes must have several connections to the tree (lines 8–9). The next step is to choose the best candidate subtree, according to the optimization criterion (line 10). If the chosen candidate subtree is better than the existing subtree, the existing subtree is removed and replaced by the candidate subtree (lines 11–13). This also updates the reduced tree. As the relay tree has improved, it is yielded to the ground operator (line 14).

The algorithm described by the pseudocode in Figure 6.16 optimizes all subtrees of T_G once, but does not return to previously optimized subtrees to optimize them again. The possible benefit of doing so is discussed in Section 6.5.2. Allowing the re-optimization of subtrees requires two changes in the algorithm. The first change is to replace the **for**-loop in line 1 with a **while**-loop that is performed as long as $J_u(T_G) \neq \emptyset$. The second change is performed if the **if**-statement in line 11 holds. In that case, all non-target key nodes of T_s' are added to $J_u(T_G)$ if they are not already in $J_u(T_G)$. The reason for this is that e.g. a non-target leaf in T_s' will be a join node in some other subtree that can potentially be improved. These changes will cause

the optimization to continue as long as there are join nodes of subtrees that have the possibility of improvement.

If any of the extensions in Section 6.3.2 were used when calculating the initial tree, then the restrictions specified by the extension must be met when optimizing the tree. Such restrictions can be handled by modifying the function Choose-best-subtree to choose a subtree that satisfies all restrictions.

Algorithm 6 improves the relay tree, and the amount of improvement depends on the initial tree. In some cases, the modified cheapest path heuristic can give trees where all subtrees treated by Algorithm 6 are already optimal and when this happens, no improvement can be made. Experimental results are available in Section 7.5.

6.5.6 Time Complexity

We will now go through the pseudocode of Algorithm 6 to determine its time complexity. The number of iterations in the for-loop is $|J_u(T_G)| = |J(T_G)|$, as this is the number of subtrees that requires optimization to optimize the complete tree. Choosing the join node $J(T_s)$ requires $O(|J_u(T_G)|) \subseteq O(|N|)$ time (line 2). Removing $J(T_s)$ from $J_u(T_G)$ is $O(1)$ (line 3). Extracting the subtree T_s is in $O(|N|)$ (line 4). Lines 5–7 are discussed below. Calculating the cost of the subtrees requires combining all reachability records in each node. Let the maximum number of terminals in a subtree be denoted by b, where $b \leq Z$. There are at most $|N|$ Pareto-optimal paths between each pair of nodes, and thus there can be at most $|N|^b$ reachability records that need to be combined. Thus the time complexity of calculating all subtrees for N nodes is in $O(|N|^{b+1})$ (line 8–9). Choosing the best subtree is $O(|N|)$. Removing T_s and inserting $T_s^{'}$ are both $O(|N|)$ (lines 12–13).

The way we choose subtrees will cause Algorithm 2 to be executed twice for each join node, once when it is join node and once as a leaf in another subtree. The only exception to this is if n_0 is a join node, in which case one less execution of Algorithm 2 is necessary as n_0 is never a leaf in any subtree. In addition, Algorithm 2 must be executed once for each terminal. In total, this requires at most $|Z|+2|J(T_G)|$ executions of Algorithm 2. This is the total number of executions of Algorithm 2 that will be performed in lines 5–7 for $|J(T_G)|$ iterations. As $|J(T_G)| \leq |Z| - 2$, this yields at most $3|Z| - 4$ executions of Algorithm 2. Thus the time complexity for executing Algorithm 2 is $O((3|Z| - 4)|N|^3) \subseteq O(|Z||N|^3)$.

This yields a total time complexity of $O(|Z||N|^3 + |N|^{b+1})$. While this time complexity may be perceived as high, the maximum number of terminals in a subtree is commonly very low and the number of Pareto-optimal paths to each node is often far less than N.

6.6 Summary

The multiple target relay problems are variants of the NP-hard Steiner tree problem. There are both continuous and discrete versions of the Steiner tree problems. Algorithms for solving the multiple target relay problems must be able to handle large three-dimensional environments with obstacles. As it is not certain that the triangle-inequality applies, no existing algorithms for the continuous Steiner problem can be used to solve our problems. Therefore, we discretize the environment and formulate our problem as a discrete Steiner tree problem.

Most discrete Steiner tree problems allow a target node to be connected to several other nodes. In the context of the relay problems, this means that a target is used to relay information. This is unrealistic and means that most Steiner tree problems are not sufficiently expressive for our needs. Upon further investigation, we found that the Steiner tree problem in a directed graph and the (partial) terminal Steiner tree problem in an undirected graph are sufficiently powerful to model the relay problems.

The algorithms for calculating a Steiner tree in a directed graph have long execution time as well as high memory consumption, making them difficult to use in our setting. The few published algorithms for the partial terminal Steiner tree problem make strong assumptions about the graph, such as that the graph is complete and that the triangle inequality holds for the cost function. Neither of these requirements is guaranteed to be fulfilled in the multiple target relay positioning problems. Therefore, the algorithms must be extensively modified to guarantee solutions to any of the multiple target relay problems, if this is at all possible. Furthermore, due to the lack of complete graphs and cost functions obeying the triangle inequality, it is unlikely that any bound on the error can be given.

Instead, modifications of existing heuristic algorithms are investigated, and it is possible to modify the cheapest path heuristic to fit our needs. Using the modified cheapest path heuristic, we are able to calculate approximate solutions to the **MTR-MinCostMinLength** and the **MTR-MinLengthMinCost** problems.

For problems involving a base station and two targets, the label-correcting algorithm from Section 5.2 is extended and used to solve all multiple target relay problems defined here. Furthermore, this algorithm is further generalized and used to improve existing relay trees, calculated by e.g. the cheapest path heuristic. The algorithm incrementally performs local optimizations of subtrees. This process can continue as long as the relay tree can be improved or for a predetermined time. The use of different optimization criteria allows that the relay tree is optimized with respect to different objectives, such as

finding the tree requiring the fewest UAVs or the cheapest tree that can be realized given a limit on the number of available UAVs.

There are algorithms for finding hop-constrained Steiner trees [97, 30] as well as the bi-criteria Steiner tree problems [65, 88]. Such algorithms could possibly be used to calculate approximate solutions to hop-constrained and bi-criteria relay trees. How much modification such algorithms require to be useful in practical relay problems is a matter for future research.

Chapter 7

Implementation and Experimental Results

The algorithms for solving the relay problems have been implemented in an existing infrastructure. The software architecture and the user interfaces are briefly described in this chapter together with the experimental results.

7.1 Software Architecture

For several years, the Artificial Intelligence and Integrated Computer Systems division [1] at the Department of Computer and Information Sciences at Linköping University has developed an experimental infrastructure for collaborative unmanned aerial systems [33, 34]. This infrastructure has several inter-linked components, such as a number of UAV systems, including two Yamaha RMAX UAVs (Figure 7.1a), several micro UAVs (Figures 7.1b–7.1d) and several software systems for experimentation. One of these systems is used for research in the area of delegation-based cooperation among UAVs.

The algorithms in this thesis, including the necessary infrastructure, has been designed and implemented as an extension to this system. This allows us to empirically test the algorithms and allows other parts of the system to use the algorithms.

The software uses a modular and distributed architecture where new services can be added through servers. Servers can be complex pieces of software such as path planners or less complex, such as an interface for manually controlling the camera onboard a UAV. For communication between the components of the system, the Common Object Request Broker

(a) Yamaha RMAX.

(b) LinkMAV [40].

(c) PingWing [27].

(d) Link Quad [93].

Figure 7.1: Some of the UAVs that are part of the experimental infrastructure for UAV research. All except the Yamaha RMAX are designed and built in-house.

Architecture (CORBA) [28] middleware is used. CORBA permits the system to be distributed over several computers. This allows both computers on the ground and on board UAVs to be used simultaneously.

The algorithms have been implemented as a *relay server* that encapsulates all algorithms and interfaces necessary to specify and solve relay problems. Figure 7.2 shows a diagram of the software architecture of the relay server.

Calls from outside the relay server go through the CORBA interface. Using this interface, problem parameters such as the positions of the base station targets and the number of UAVs are set, and the values are stored internally. From the interface, the appropriate function for calculating relay chains and trees can be called, and the result is returned through the CORBA interface. Missions can be specified from the Graphical User Interface (GUI), which then calls the surveillance mission server that delegates tasks to specific UAVs. The delegation is very briefly described in the use case below. The CORBA interface can also be called from other servers in

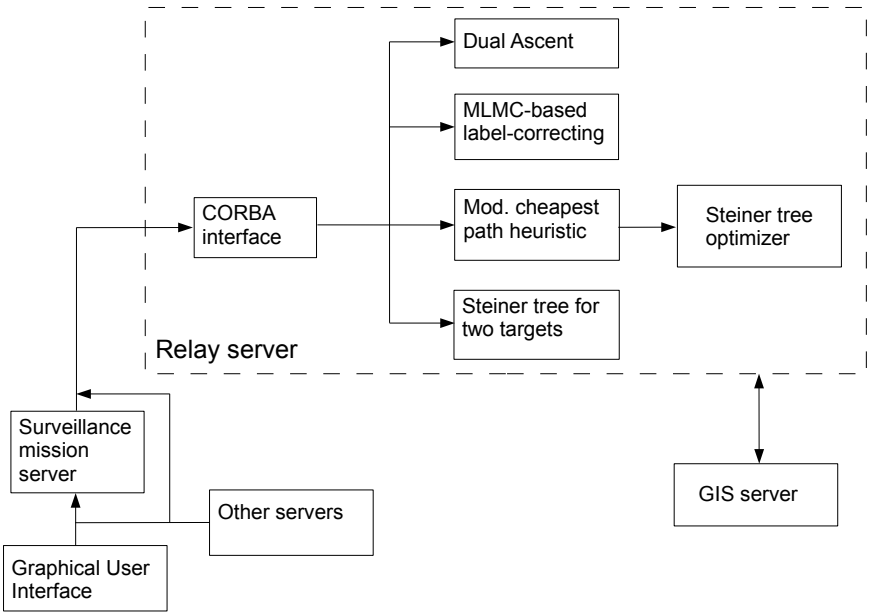


Figure 7.2: The major components in the relay server (dashed rectangle) and the main connections to other parts of the UAV infrastructure.

the system.

The GUI allows users to set problem-specific parameters such as the communication range, the number of allowed UAVs and more. It also has a two-dimensional visualizer that displays the environment from a top-down perspective. Figure 7.3 shows the interface after solving a single target relay problem. In the visualizer, the base station is marked by a yellow square near the bottom left corner, and the target is marked by a red circle. Each relay chain is visualized in a distinct color, and the position of each UAV is marked by a circle. This is the common GUI when relay problems are specified and solved, but there is also an optional three-dimensional visualizer available (Figure 7.4 on page 108).

Other information about e.g. the environment that is required to perform calculations of relay chains or trees is retrieved from other servers in the system, most often the GIS server, which stores information about the environment. With the required information available, the graph is created internally in the relay server.

As described earlier, the algorithms and the interface presented here are part of a infrastructure, which is typically operated by one or more human ground operators. The system allows the ground operator to instantiate a variety of missions, such as traffic monitoring, photogrammetry and surveil-

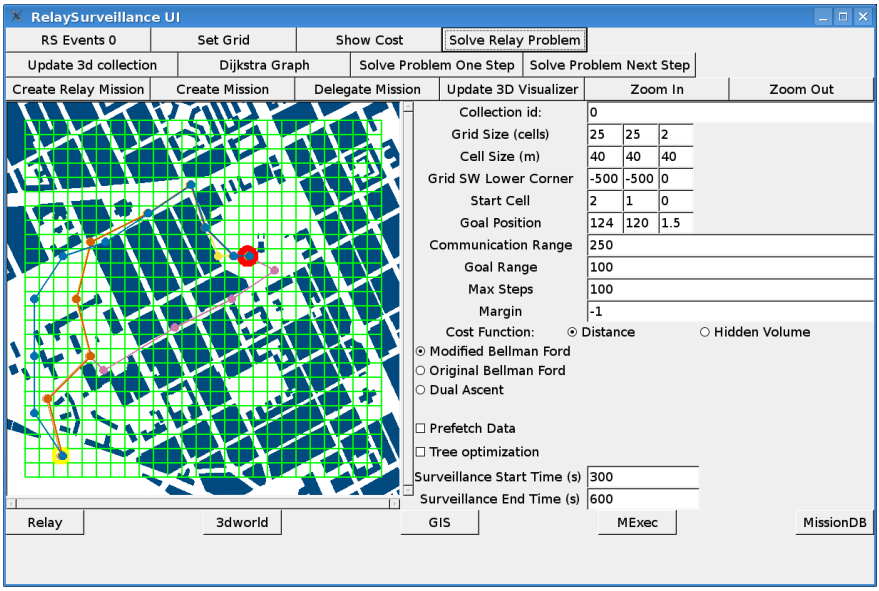


Figure 7.3: Screen shot from the graphical user interface after solving a single target relay problem. The map in the left part of in the interface displays four different relay chains between the base station in the lower left hand corner and the target in the middle.

lance. Here we describe a use case where the users set up a surveillance mission.

When a surveillance mission is in the planning stage, the ground operator sets problem parameters such as the position of the base station and one or more targets, reachability and cost functions as well as discretization. Depending on whether there is a single target or if there are several targets, different options for choosing algorithms are presented.

The ground operator also decides the objective of the calculations. For both single and multiple target surveillance, he can choose whether to minimize the number of UAVs required for surveillance to find the minimum cost relay chain/tree or find a chain/tree of minimal cost. For single target problems, the option to find several different relay chains, i.e. all Pareto-optimal chains, is also available. For multiple target problems, he can also choose whether to continue to optimize the relay tree after the initial tree has been calculated.

In the current prototype system, a broadcast is issued with the intention of finding a set of UAVs that are available for performing a relay mission. The UAVs in the area respond to the request for participation. Naturally, if some UAVs already have a mission or are unable to participate in a relay

mission for other reasons, such as lacking appropriate sensors or communication equipment, they respond that they are unable to participate. UAVs able to participate preliminarily accept the request for participation.

At this point all necessary problem parameters have been set and the appropriate algorithm is used to calculate the solution. Once the solution has been calculated, it is displayed to the ground operator. If the solution cannot be realized using the available number of UAVs, more UAVs must be found or some parameter must be changed to find a solution requiring fewer UAVs.

Otherwise, a task specification tree [35] is created corresponding to the selected relay chain or tree. The surveillance mission service then calls the delegation system [58], which attempts to delegate this tree to an appropriate set of UAVs supporting the required roles. If all constraints associated with the mission can be satisfied, including timing constraints, the UAVs accept their tasks. When a sufficient number of UAVs can take part in the mission, the ground operator accepts the final delegation of roles and restrictions.

Once the delegation process is finished and each UAV has been assigned a role in the mission, each UAV uses its own path planner to find a flyable trajectory to its designated position. Due to the decentralized nature of the path planning step, it can be done in parallel. Now everything is set up for executing the mission and the surveillance can begin as soon as all UAVs have arrived at their positions.

7.2 Problem Setup for Empirical Testing

For testing, we used directed graphs constructed from grids and several different environments. All environments have the same size, 1000 $\times$ 1000 $\times$ 80 meters. The grid cell size was varied between 10 and 40 meters. The resolution in the horizontal direction has a greater impact on the probability of finding good paths, and this is reflected in the choice of grid cell sizes. All testing was performed on a standard PC with a 2.4 GHz Core 2 Duo CPU and 2 GB RAM.

For all testing, the testing used reachability functions based on free line-of-sight and a range of 100 meters for both communication and surveillance. Two different cost functions were used: one based on distance and the other on obstructed volume. The distance cost function has a constant cost of 300 up to 60 meters, corresponding to the assumption that communication within this distance will have comparatively constant, but not perfect, quality. After 60 meters, the cost increases with the square of the distance. This tests the case where a wide variety of Pareto-optimal relay

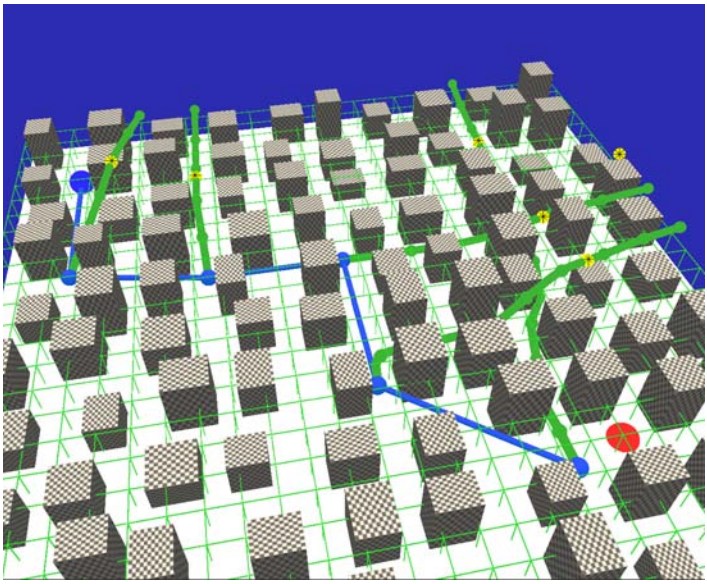


Figure 7.4: Randomized urban environment.

chains is generated for any target. The second cost function is based on obstructed volume (see Section 3.3.2). This generally results in considerably fewer Pareto-optimal chains, testing the performance of the algorithms for this end of the spectrum as well.

Randomized Urban. The first environment used in testing is an urban environment with semi-random placement of 100 tall buildings, as shown in Figure 7.4. To reduce clutter, the figure displays a sparse discretization and only the "lowest" level of grid cells. In this figure, a specific relay chain is visualized. The base station is in the upper left corner and is connected by dark lines representing communication links to dark spheres denoting intended positions for relay and surveillance UAVs. The target is in the lower right corner and is visible from the last UAV in the chain. Subtasks have been delegated to several UAVs, marked by a dark stars on lighter spheres. In the figure, the simulated UAVs have used their path planners to generate individual flight paths (indicated by lighter lines) and are in the process of flying to their intended positions.

Urban With Boulevards. The second simulated environment randomizes the buildings roughly in four blocks, and leaves space for two broad boulevards that cross in the middle. The buildings in this environment are

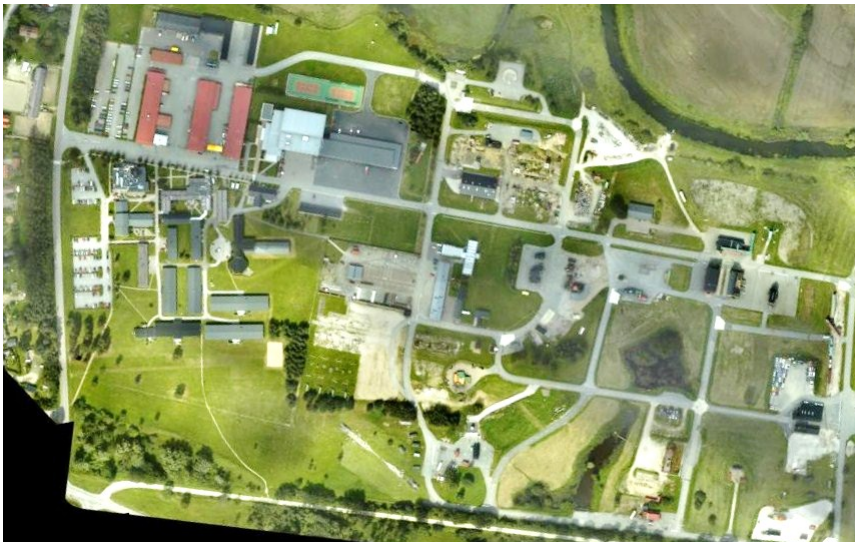


Figure 7.5: The Revinge emergency services training ground.

more diverse in size and may intersect each other, creating an urban environment where buildings share common walls.

Randomized Dense Urban. The third environment also places buildings semi-randomly. However, it generates 625 smaller buildings, creating an environment where each node has fewer neighbors.

Stockholm. The Stockholm environment is an area of central Stockholm extracted from OpenStreetMap [78]. In this environment, the buildings form city blocks of irregular shapes and different sizes. The many houses make this a very dense area, and therefore, considerably fewer nodes are created, compared to the other environments. As the map does not contain any building heights, all buildings are considered infinitely high when testing.

Figure 7.3 shows a part of the Stockholm environment.

Revinge. The Revinge environment is a 3D model of an emergency services training facility in the south of Sweden (Figure 7.5). It predominantly consists of open areas with some buildings scattered throughout the environment.

Properties of the Test Environments. Some information about the graphs is displayed in Table 7.1. The average number of nodes and edges are

shown for each world and discretization. Similarly, the minimum, average and maximum node degrees are also displayed. As expected, the average and maximum node degree increases with decreasing grid cell size, as there are more nodes for which the reachability functions hold. The numbers are for testing of the single target relay problems. The numbers for the multiple target problems are slightly higher as there are several target nodes and edges connecting these nodes to the rest of the graph.

Recall that that k^*_{max} is the maximum number of hops required in an MLMC-tree, i.e. no minimum cost path will require more than k^*_{max} hops. The value of k^*_{max} is commonly larger for the cost function based on distance than the cost function based on obstructed volume. For the cost function based on distance, the cost increases faster than linearly. Therefore is it possible to decrease the cost of a path by exchanging a long edge for several shorter edges. This allows a larger set of Pareto-optimal paths and increases the value of k^*_{max}.

The values of k^*_{max} in the dense urban and the Stockholm environments are considerably higher than for the other environments. This is especially evident for the coarser discretizations. Due to the low node connectivity, relay chains are in some cases forced to take long detours to reach their targets. The values of k^*_{max} in Table 7.1 are the average of k^*_{max} for the different environments and discretizations and are from the testing calculations of Pareto-optimal relay chains.

All algorithms for single target relay problems tested here calculate paths to all reachable nodes from the base station. Therefore, Table 7.1 includes the number of reachable nodes and edges only. There can be small graphs that are not reachable from the main graph due to obstacles.

7.3 Pareto-Optimal Relay Chains

To solve **STR-ParetoLimited** problems, the new MLMC-tree-based label-correcting algorithm (Algorithm 2) was used, and for comparison, we used the truncated version of Bellman-Ford (Algorithm 1). Testing was performed in the environments described above, and in each environment, we used the seven discretizations described in Table 7.1. For each discretization, we randomly generated 100 combinations of start and goal positions, and to ensure that all Pareto-optimal chains were found, $M = \infty$ was used.

Figures 7.6 and 7.7 display the average execution times for generating all Pareto-optimal chains, using cost function based on obstructed volume and distance, respectively. Times for Algorithm 1 are displayed in black and times for Algorithm 2 are in green. The standard deviation in execution time is indicated using error bars.

World	Cell size (meters)	$\|N\|$	$\|E\|$	Node degree Min/avg/max	Avg. Vol.	k^*_{max} Dist.
Randomized urban	40×40×40	1,022	21,815	5/21/35	14.1	23.6
	33×33×40	1,791	53,767	6/30 /47	14.9	24.1
	25×25×25	3,796	257,711	11/67/113	13.9	20.1
	20×20×20	7,846	1,102,063	21/140/221	13.5	19.0
	15×15×20	13,852	3,741,793	30/270/434	12.4	17.7
	12×12×20	22,008	9,286,019	40/421/680	12.5	18.1
	10×10×20	31,807	18,753,049	61/589/948	12.3	17.8
Urban with boulevards	40×40×40	989	27,455	1/27/42	12.7	22.1
	33×33×40	1,781	62,547	1/35/50	12.6	23.2
	25×25×25	3,775	339,148	2/89/135	12.0	18.4
	20×20×20	7,941	1,456,997	7/183/278	13.3	17.6
	15×15×20	13,813	4,850,503	11/351/531	12.8	16.9
	12×12×20	21,885	11,970,877	11/546/831	14.4	18.1
	10×10×20	31,965	24,452,326	15/764/1,154	12.7	17.4
Randomized dense urban	40×40×40	660	3,176	1/4/15	27.0	41.2
	33×33×40	1,493	23,888	1/16/42	19.3	24.7
	25×25×25	2,975	62,134	2/20/57	16.8	22.7
	20×20×20	4,107	167,347	3/40/99	15.1	20.2
	15×15×20	10,852	876,771	7/80/220	13.3	18.1
	12×12×20	17,325	2,206,373	11/127/373	13.2	18.7
	10×10×20	25,235	4,545,123	15/180/510	13.0	18.0
Stockholm	40×40×40	467	7,046	1/15/37	21.2	34.0
	33×33×40	976	16,538	3/16/47	22.7	25.4
	25×25×25	2,263	104,242	1/46/130	19.5	25.8
	20×20×20	4,860	455,256	1/92/261	17.4	22.7
	15×15×20	8,707	1,496,489	1/171/498	16.7	20.7
	12×12×20	13,647	3,691,925	1/270/775	15.3	20.2
	10×10×20	20,003	7,581,444	1/379/1,085	15.1	20.0
Revinge	40×40×40	1,170	38,903	1/33/42	13.9	23.0
	33×33×40	2,049	86,351	1/42/51	13.9	24.0
	25×25×25	4,480	479,167	2/106/137	13.7	20.0
	20×20×20	9,341	2,047,203	7/219/280	13.4	18.5
	15×15×20	16,273	6,784,873	11/416/532	12.3	17.3
	12×12×20	25,663	16,762,197	1/653/832	12.4	18.0
	10×10×20	37,307	33,976,220	1/910/1,157	12.3	17.5

Table 7.1: Information about worlds and discretizations in empirical testing of algorithms for single target relay problems.

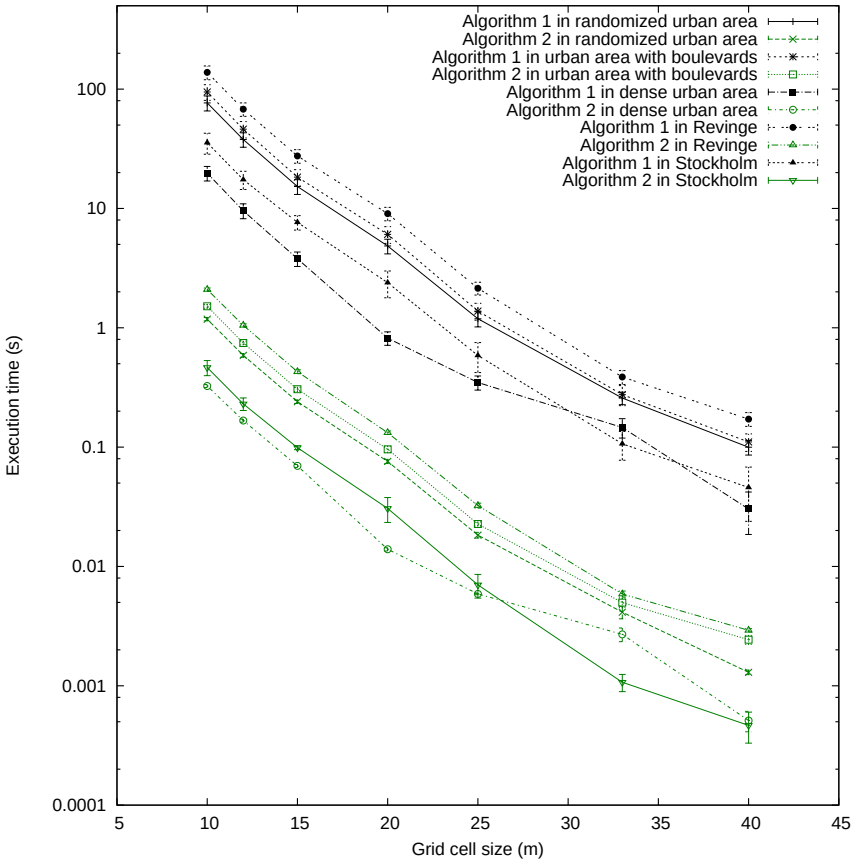


Figure 7.6: Test results for testing of algorithms for the **STR-Pareto-Limited** problem, with cost function based on obstructed volume.

From Figures 7.6 – 7.7 it is evident that Algorithm 2 is able to decrease the execution time more when the cost function based on obstructed volume is used. This cost function commonly generates few different Pareto-optimal chains. After this set of chains is found, no calculations are necessary and thus this cost function offers great potential for improving the execution time. The speedups are in the order of 45–75 times except in the Stockholm environment where the speedups are 75–100 times.

The cost function based on distance (Figure 7.7) allows for a greater set of Pareto-optimal chains, often 4–10 chains. Many more calculations need to be performed and the calculations performed by Algorithm 2 cannot be terminated as early. Even in these more difficult circumstances, Algorithm 2 achieves a speedup in the order of 10–22 in the Stockholm environment. For

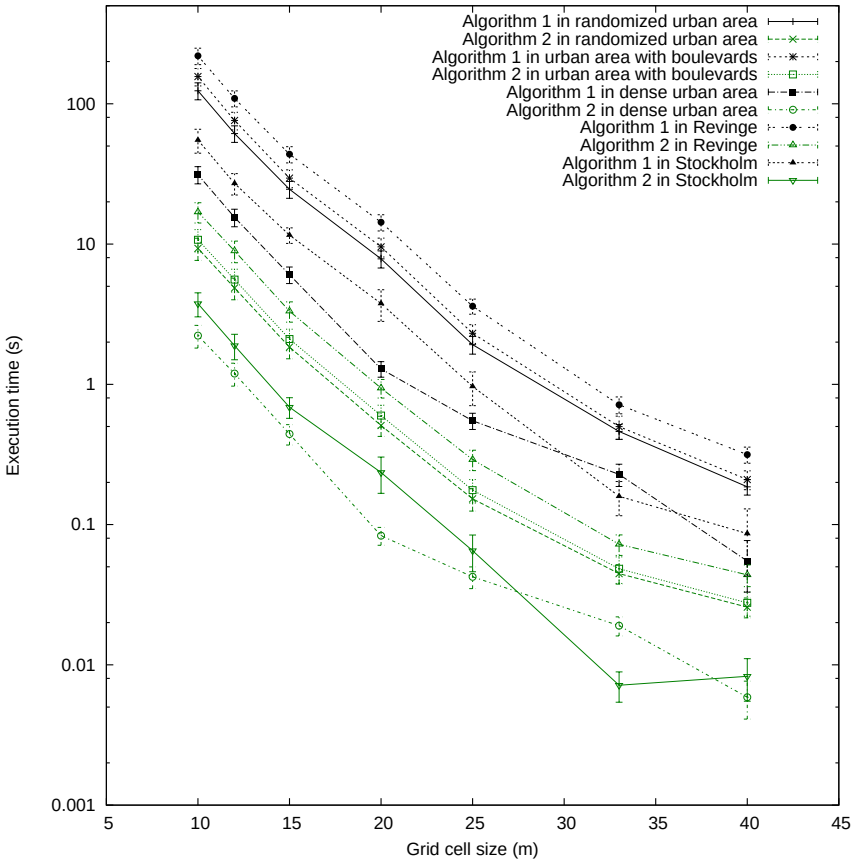


Figure 7.7: Test results for testing of algorithms for the **STR-Pareto-Limited** problem, with cost function based on distance.

the other environments the speedup is 7–10 for the two coarsest discretizations (33 and 40 m grid cell size) and 12–16 for the other.

When run in dense urban environments, the execution times of the algorithms vary more depending on the discretization, for both cost functions. For Algorithm 2 in the Stockholm environment, the execution time for test cases in the discretization with 33 m grid cells is lower than the execution time for the discretization with 40 m grid cells. This is due to fewer solutions being created for the finer discretization. On average, the number of solutions is two for the discretization with 33 m grid cells and six for the discretization with 40 m grid cells. The smaller number of solutions can also be seen for Algorithm 1, where the increase in execution time from 40 m to 33 m grid cells is quite small. For both algorithms, the dense Stockholm

environment also causes a greater standard deviation of the execution times, as compared to the other environments.

The conclusion of testing of algorithms for the **STR-ParetoLimited** problem is that our new algorithm (Algorithm 2) offers far better performance in the tested environments. For the cost function based on distance, the speedup is less than for the cost function based on obstructed volume, as the number of Pareto-optimal chains typically is larger. However, Algorithm 2 consistently outperforms Algorithm 1, in many cases by an order of magnitude.

7.4 Optimal Chains Using At Most M UAVs

To test the performance of the dual ascent algorithm (Algorithm 3) for solving the **STR-MinCostLimited** problems, the same environments as for **STR-ParetoLimited** were used. Both Algorithm 1 and Algorithm 2 are able to solve this problem, but as Algorithm 2 is the fastest algorithm available, it is used for comparison. We ran separate tests for different values of M, simulating a user requesting relay chains fo different lengths.

For the **STR-MinCostLimited** problem, the more interesting test cases occur when the cost function based on distance is used, as it often produces several Pareto-optimal chains. For this reason, this cost function was used in testing. For each discretization, we used randomly generated 100 combinations of start and goal positions.

The test results are available in Table 7.2, in which the first columns display information about the discretizations. The last four columns display, for each algorithm, the percentage of test cases where one of the tested algorithms is faster than the other, and the average speedup when this happens. Only test cases with a speedup larger than 1% are displayed here. For test cases with a smaller speedup, the algorithms are considered equally fast. The numbers in the table represent the test cases with a speedup greater than 1%.

Figure 7.8 exemplifies the difference in the order in which the paths are calculated. Times for the MLMC-based label-correcting algorithm (Algorithm 2) are in green and times for the dual ascent algorithm (Algorithm 3) are in blue. Algorithm 2 first calculates the longest (length 10) and cheapest chain during preprocessing, and all remaining chains are calculated in order of increasing length. Thus, the maximum execution time is for the chain of length 9. Algorithm 3 also finds the longest path first, and then goes on to find incrementally shorter paths. In general, longer paths can be generated quite quickly as they require fewer iterations. The chain of length 5 was found in shorter time than the chain of length 6 as separate tests

World	Cell size (meters)	Algorithm 2 faster		Algorithm 3 faster	
		% paths	Speedup	% paths	Speedup
Randomized urban	40×40×40	0	0	100	2.96
	33×33×40	2	1.09	98	2.34
	25×25×25	26	1.48	74	2.03
	20×20×20	48	3.28	52	1.83
	15×15×20	84	11.6	16	1.63
	12×12×20	87	25.5	13	2.18
	10×10×20	90	33.9	10	2.01
Urban with boulevards	40×40×40	0	1.01	100	3.16
	33×33×40	1	1.04	99	2.46
	25×25×25	22	1.32	78	2.03
	20×20×20	49	2.87	51	1.85
	15×15×20	82	8.42	18	1.69
	12×12×20	74	23.7	26	1.51
Randomized dense urban	40×40×40	3	1.02	97	3.43
	33×33×40	1	1.09	99	2.70
	25×25×25	7	1.28	93	1.95
	20×20×20	34	1.68	66	1.83
	15×15×20	70	6.39	30	1.61
	12×12×20	80	16.4	20	1.98
	10×10×20	85	18.3	15	1.91
Stockholm	40×40×40	2	1.06	98	3.18
	33×33×40	5	1.08	95	2.56
	25×25×25	7	1.54	93	2.60
	20×20×20	32	1.74	68	2.08
	15×15×20	57	4.81	43	2.03
	12×12×20	72	13.5	28	1.87
	10×10×20	79	18.7	21	1.94
Revinge	40×40×40	1	1.01	99	2.17
	33×33×40	7	1.13	93	1.57
	25×25×25	54	1.43	46	1.35
	20×20×20	92	2.78	8	1.28
	15×15×20	97	11.1	3	1.22
	12×12×20	98	24.1	2	1.19
	10×10×20	99	28.0	1	1.40

Table 7.2: Speedup and test case information for testing of the **STR-MinCostLimited** problem.

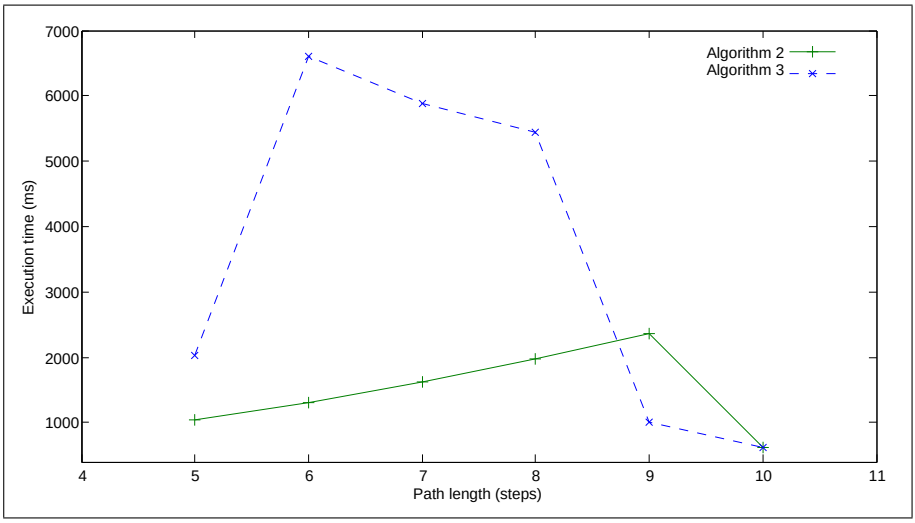


Figure 7.8: Timing results for Algorithm 2 and Algorithm 3 for a specific test case, showing the times for when chains of specific lengths are found. Data from urban environment with $12 \times 12 \times 20$ grid cells.

were run for different values of M, and the search space was more effectively constrained when calculating the shorter chain.

We now give three examples of how the execution time is affected by the discretization in the Stockholm environment. Figure 7.9 displays the average execution times for a discretization with $40 \times 40 \times 40$ m grid cells. As the lengths of the longest and cheapest chains vary, relative path lengths have been used. As an example, assume that the longest chain has a length of 20 hops. Then, the time reported for a relative path length –5 corresponds to a path length of 20–5=15 hops. Naturally, not all test cases have the same number of Pareto-optimal chains and this affects the shape of the curves for execution times. For example, the execution times for relative length –10 are averaged over considerably fewer (and on average more difficult) test cases than the results of relative length 0. For this reason, the curves at that end of the spectrum have a more uneven appearance.

Figure 7.10 displays execution times for the discretization with $20 \times 20 \times 20$ m grid cells. Algorithm 3 is faster for paths slightly shorter than the cheapest path, and then the gap between the times for Algorithm 2 and Algorithm 3 increases as shorter paths are desired. For some path lengths, e.g. -14 to -5, Algorithm 2 is the faster as Algorithm 3 must perform a large number of iterations. Although each iteration is quite fast due to the constrained search space, Algorithm 2 is still faster.

Figure 7.11 displays execution times for the discretization with $10 \times 10 \times 20$

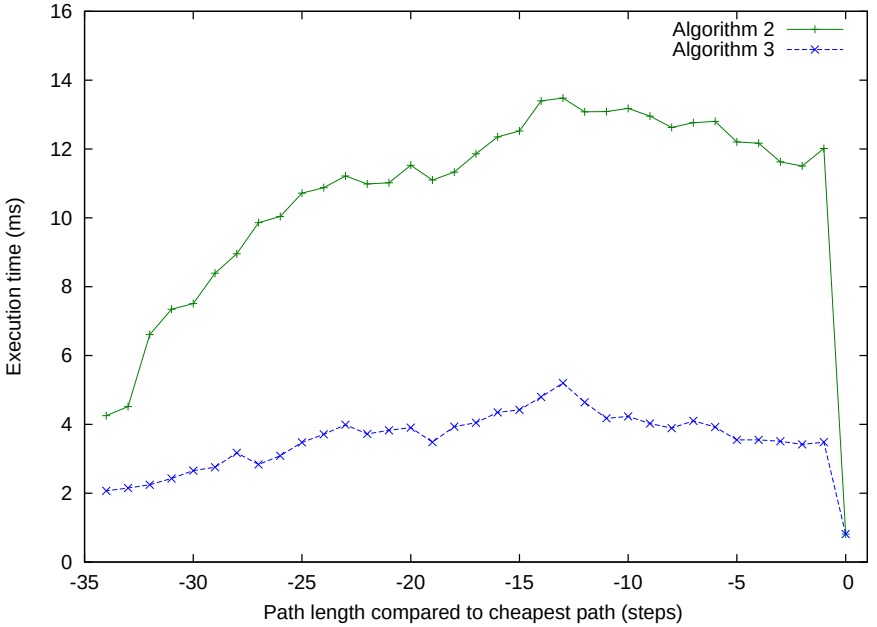


Figure 7.9: Timing results for testing of algorithms for the **STR-Pareto-Limited** problem in the Stockholm environment with 40×40×40 m grid cells.

m grid cells. Due to the large difference in execution time, this figure uses a log-scale. It is easy to see that Algorithm 2 provides a more consistent performance than Algorithm 3 in this environment. Here the pruning of the search space that Algorithm 3 performs is not sufficient to yield good performance. In many of the iterations, edges that do not affect the path from n_0 to τ_1 get included in the tree, and therefore a large number of iterations must be executed to find a shorter path. The greater number of iteration of course leads to a longer execution time.

In graphs with few nodes and edges, the dual ascent algorithm outperforms Algorithm 2, in many cases being 2–3 times faster. As the number of nodes and edges increase, Algorithm 2 offers better performance than Algorithm 3. Although Algorithm 3 constrains the search space, this is not always enough as the number of calculations of MLMC-trees is quite large in some cases. For example, consider the Stockholm environment, in which the dual ascent algorithm offers the better performance in a majority of the cases, for the four coarsest discretizations. As the number of nodes and edges increase, Algorithm 2 offers much better performance, and the same trend is evident for the other testing environments.

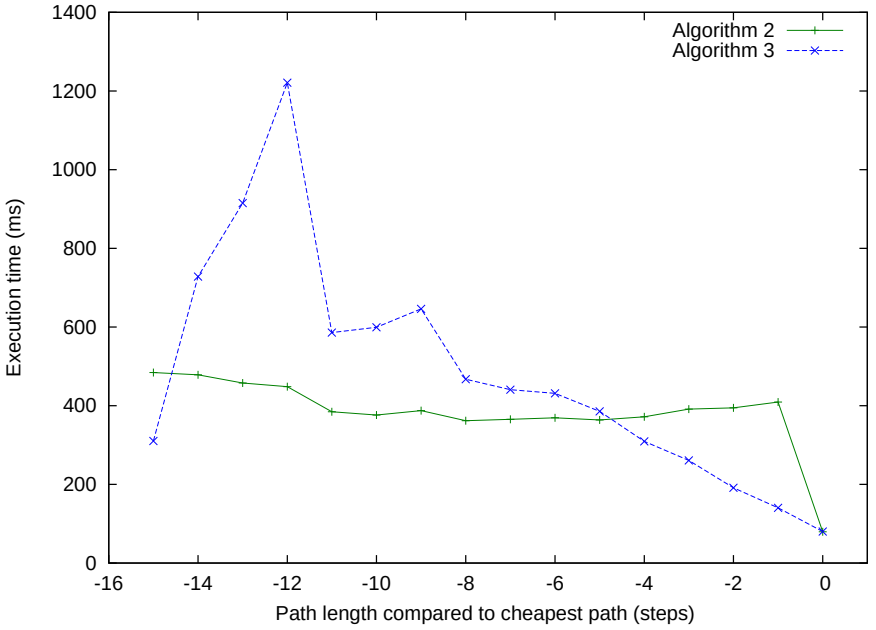


Figure 7.10: Timing results for the **STR-ParetoLimited** problem in the Stockholm environment with 20×20×20 m grid cells.

From the test results is the evident that the dual ascent algorithm is more suitable in dense environments, where the number of nodes and edges is low, and Algorithm 2 is better suited for environments where there are many nodes and edges.

7.5 Relay Trees

To test the algorithms for multiple target relay positioning problems, we used the same environments and discretization as earlier, but this time we used nine targets distributed in three clusters with three targets in each cluster. Each randomized cluster had to be at least 300 meters from the base station and within each cluster, the targets were at most 100 m from each other. The cost function based on distance, as described in the beginning of this chapter, was used.

The initial relay trees were generated using Algorithm 4. We ran separate tests for the **MTR-MinLengthMinCost** and the **MTR-MinCost-MinLength** problems. For the former, we used compound costs (see page 41) of the form $\langle c, l \rangle$, i.e. first the cost was minimized and in case of equal cost of several paths, the shortest path was chosen. The latter problem used the

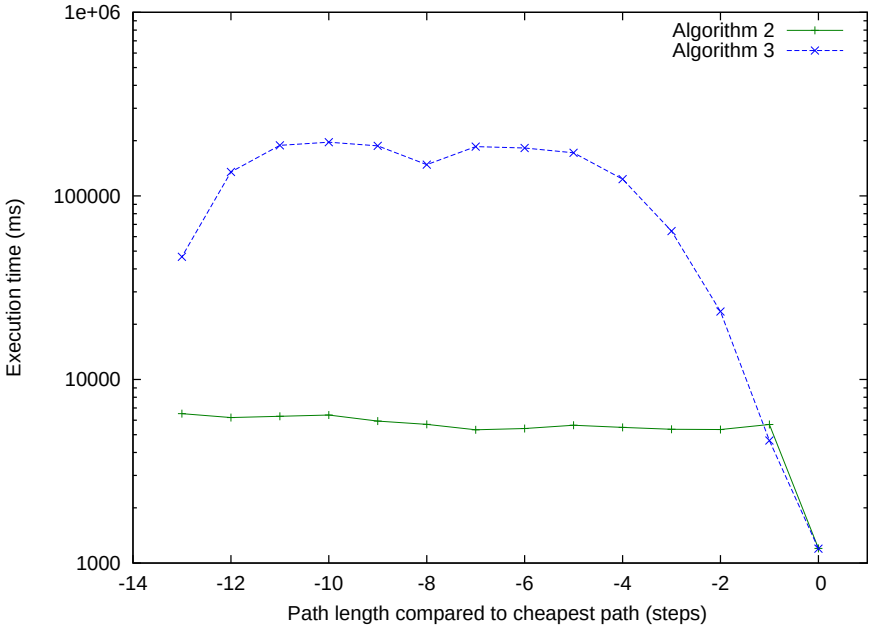


Figure 7.11: Timing results for testing of algorithms for the **STR-Pareto-Limited** problem in the Stockholm environment with 10×10×20 m grid cells.

reversed prioritization. Execution times for **MTR-MinLengthMinCost** in the different environments are displayed in Figure 7.12 and execution times for **MTR-MinCostMinLength** are displayed in Figure 7.13. From the figures, we can see that execution times for the both problems are quite similar, and that there is large difference in calculation time for the different environments. There is sometimes an order of magnitude difference for the different environments but the same discretization. The reason is that the size of the graphs, especially the number of edges, differ considerably in the different environments. A smaller graph gives fewer possibilities when calculating relay trees, which leads to a shorter execution time.

After the initial tree was calculated, we used Algorithm 6 to improve the trees further. The algorithm was set to first optimize subtrees further away from the root node, and then subtrees progressively closer to the root node were chosen. Optimization was performed until no improved subtree could be calculated. The execution times for Algorithm 6 for the **MTR-MinLengthMinCost** problem are available in Figure 7.14 and times for the **MTR-MinCostLimited** problem are available in Figure 7.15.

The average improvements of the primary and secondary factors (UAVs

and cost for the **MTR-MinCostMinLength** problem and cost and UAVs for the **MTR-MinLengthMinCost** problem), are displayed in the rightmost part of Table 7.3. The numbers are the improvement in percent over the initial tree.

The improvement in the number of UAVs is often in the 5–8% range, and the cost can often be decreased by more than 5%. When focusing on improving the cost, the improvement is often in the 4–6% range and the improvement in the secondary factor (number of UAVs) is slightly lower. From this, it is evident that focusing primarily on improving the number of UAVs in the tree yields a greater improvement as compared to focusing primarily on decreasing the cost. Also, the improvement is greater in larger

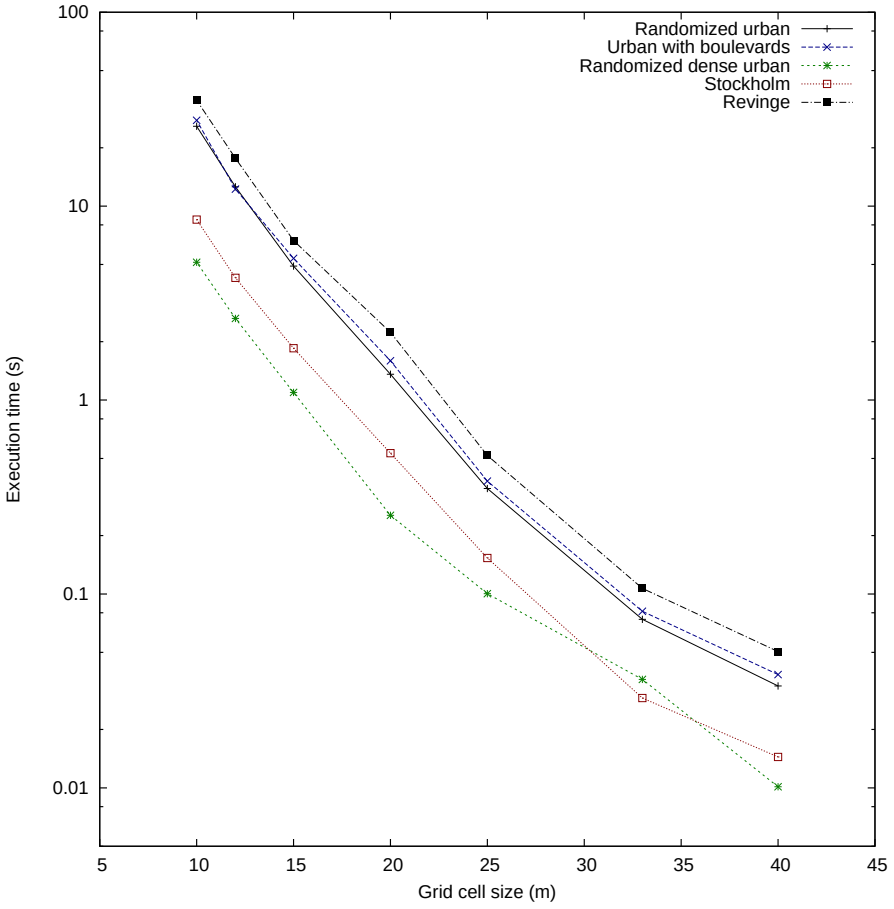


Figure 7.12: Execution times for Algorithm 4 for calculating the initial relay tree for the **MTR-MinLengthMinCost** problem.

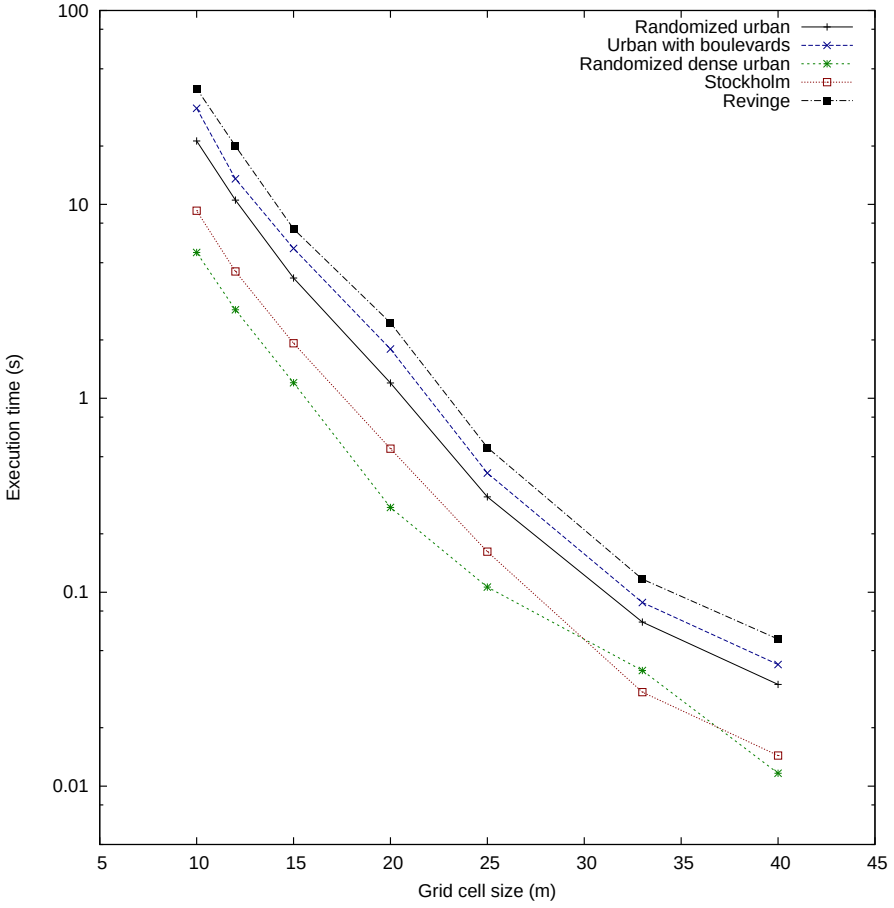


Figure 7.13: Execution times for calculating the initial relay tree using Algorithm 4, for the **MTR-MinCostMinLength** problem.

graphs, which is to be expected as there are more opportunities to improve an existing subtree due to having more nodes to choose from.

When interpreting these numbers, it is important to remember that the cheapest path heuristic has proven to be competitive with more advanced heuristics in directed graphs [50, 96]. In the testing performed by Hsieh et al. [50], all results were within 5.5% from the optimal cost. While the graphs were much smaller than the ones used here, it is an indication that the cheapest path heuristic often finds high-quality solutions.

Figure 7.16a shows the distribution of cost improvement for the Revinge environment with 20×20×20 m grid cells. The y-axis displays the number of test cases that achieved a given improvement. Although the average improvement was a little more than 5%, one tree was improved by more

World	Cell size (meters)	Avg. improvement, % MCML UAVs	MCML Cost	MLMC Cost	MLMC UAVs
Randomized urban	40×40×40	6.3	8.1	3.2	2.9
	33×33×40	7.0	1.3	4.8	4.9
	25×25×25	7.0	9.2	4.4	3.7
	20×20×20	7.0	8.8	5.2	4.8
	15×15×20	7.7	5.0	5.9	5.9
	12×12×20	8.7	6.3	5.8	5.9
	10×10×20	9.0	5.8	6.2	6.0
Urban with boulevards	40×40×40	6.9	6.5	3.7	3.0
	33×33×40	6.1	4.3	4.8	4.8
	25×25×25	6.2	10.1	4.7	4.0
	20×20×20	7.4	8.4	4.9	4.5
	15×15×20	7.9	4.4	5.8	5.6
	12×12×20	8.3	4.1	6.3	6.3
	10×10×20	7.9	8.5	5.9	5.1
Randomized dense urban	40×40×40	3.0	2.7	2.1	0.6
	33×33×40	5.1	3.1	3.5	3.7
	25×25×25	6.6	4.5	3.9	3.6
	20×20×20	6.9	4.4	4.6	5.6
	15×15×20	9.7	2.9	5.4	6.4
	12×12×20	8.7	4.7	5.2	5.8
	10×10×20	10.1	6.9	6.3	7.8
Stockholm	40×40×40	4.2	5.3	2.0	0.5
	33×33×40	3.9	5.6	3.3	2.6
	25×25×25	3.7	6.0	3.4	3.4
	20×20×20	5.1	5.5	3.7	3.1
	15×15×20	6.2	6.4	4.7	4.6
	12×12×20	6.6	3.9	4.5	4.3
	10×10×20	6.5	3.0	4.4	4.8
Revinge	40×40×40	7.8	8.2	4.0	3.8
	33×33×40	7.0	3.8	5.0	4.6
	25×25×25	6.7	6.9	4.7	3.1
	20×20×20	6.3	9.1	5.3	4.6
	15×15×20	7.3	6.5	5.0	4.9
	12×12×20	8.2	4.1	5.4	4.7
	10×10×20	8.2	4.9	6.1	5.2

Table 7.3: Improvement in primarily the number of non-terminal nodes (UAVs) and secondarily the tree cost for the **MTR-MinCostMinLength** and the reverse prioritization for the **MTR-MinLengthMinCost** problem, as calculated by Algorithm 6.

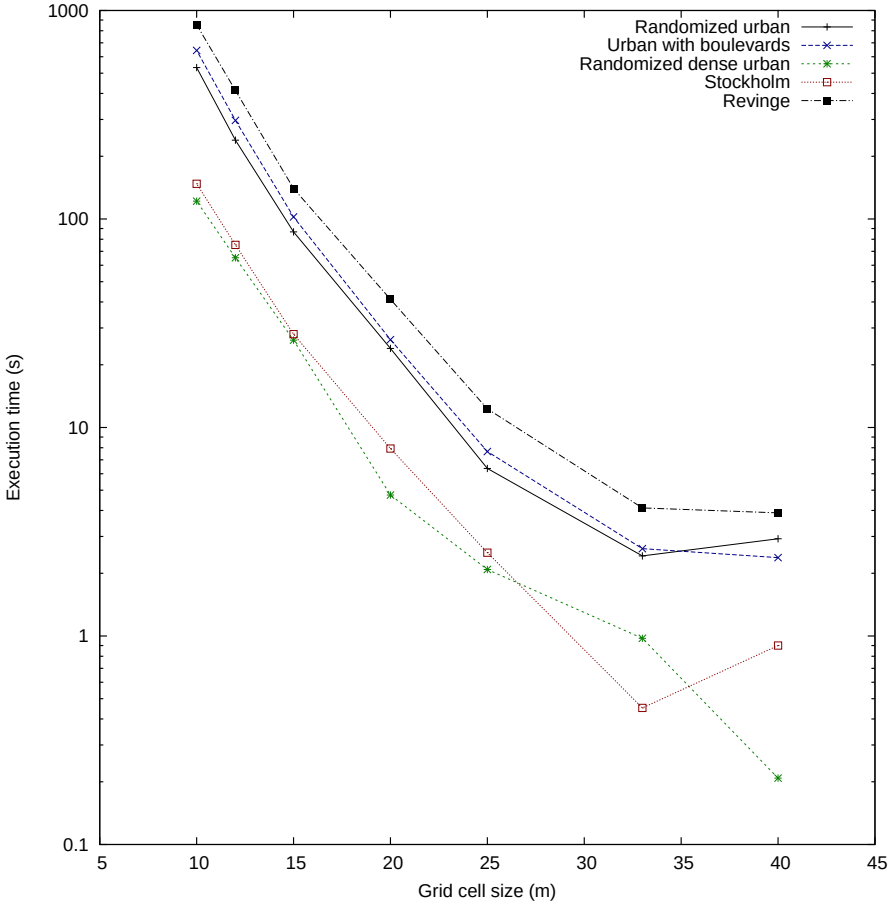


Figure 7.14: Execution times for Algorithm 6 for optimizing the relay tree, for the **MTR-MinLengthMinCost** problem.

than 13%. Figure 7.16b displays the improvement in the number of UAVs in the tree for the same test case. The average improvement in the number of UAVs is less than the improvement of the tree cost, but the maximum improvement is larger.

To conclude we give an example for a certain test case from the randomized urban environment with grid cells measuring 33×33×40 meters. Here Algorithm 6 is used to find the cheapest feasible tree. The original tree was calculated using Algorithm 4. Figure 7.17 shows how the number of UAVs and the tree cost change during optimization. The solid black curve is the number of UAVs in the tree and the dashed green curve is the tree's cost. The ground operator has set the number of UAVs available $M = 23$. As the first tree uses 26 UAVs, it is infeasible and the optimization criterion

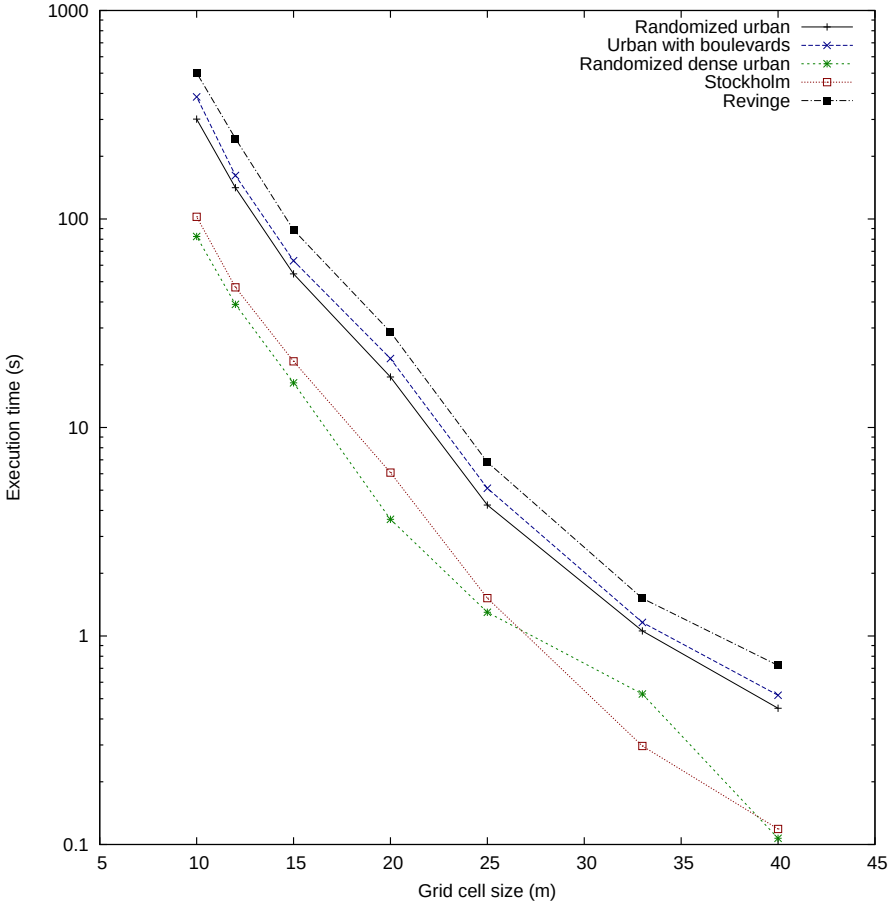
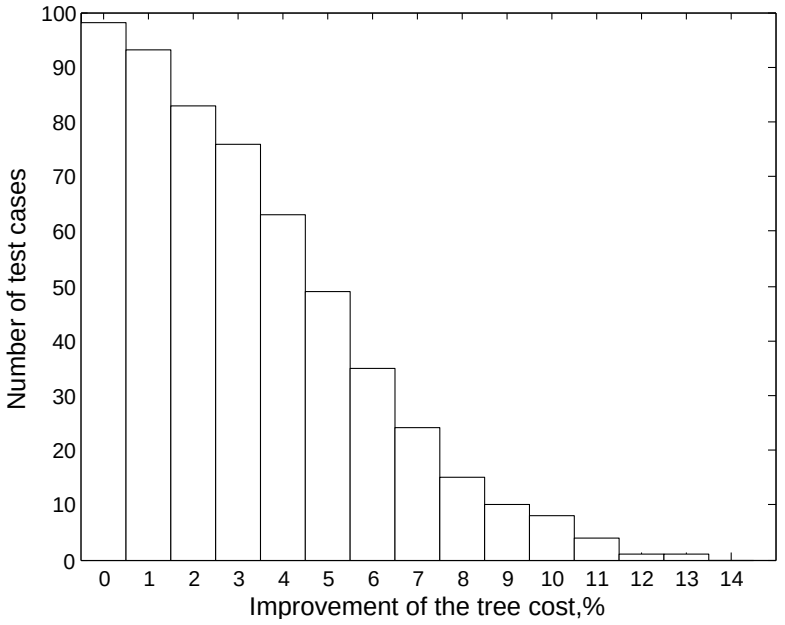
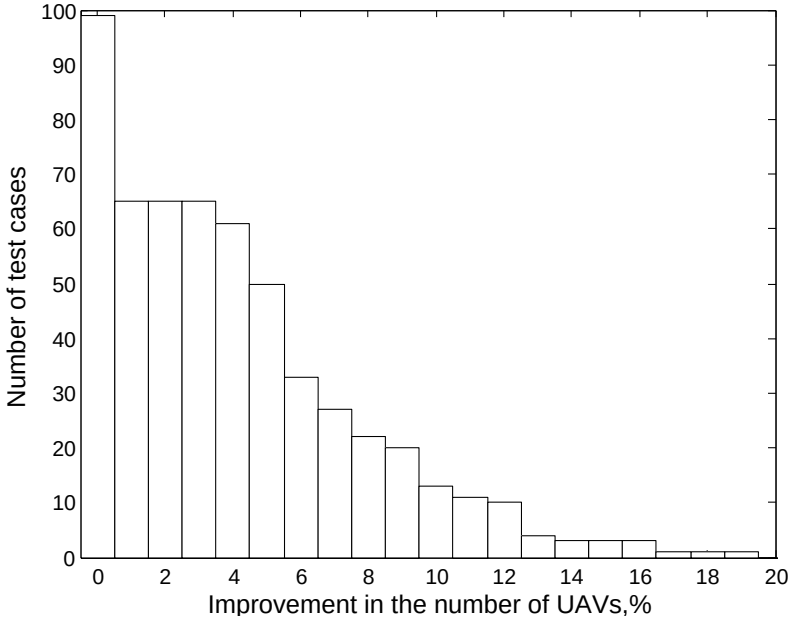


Figure 7.15: Execution times for optimizing the relay tree using Algorithm 6, for the **MTR-MinCostMinLength** problem.

is to minimize the number of UAVs in the tree. The second successful optimization finds a tree using 21 UAVs. The optimization criterion is then automatically changed to finding the least cost feasible tree, i.e. a tree using at most 23 UAVs. Each new tree decreases the cost, until no lower cost tree is found, after the ninth subtree optimization. In total, the number of UAVs is decreased from 26 to 23 and the cost is only marginally increased. The complete optimization took less than 10 seconds and each improved tree is available to the user as soon as it is calculated.



(a)



(b)

Figure 7.16: Data for the Revinge environment with 20×20×20 m grid cells. The y-axis displays the number of test cases with a given improvement.

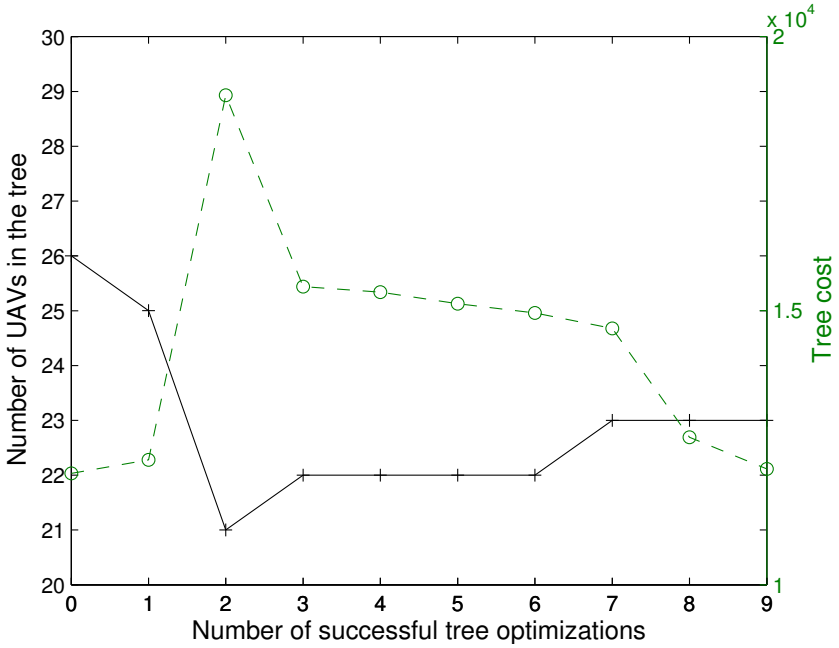


Figure 7.17: Minimization of the number of hops is performed until a feasible tree is found, using at most 23 UAVs. Once a feasible tree is found, the optimization criterion is changed to finding the least cost feasible tree.

Chapter 8

Discussion

In this thesis, we have provided definitions for relay positioning problems for both single and multiple targets. We have also suggested a solution process that enables us to model such problems with a large degree of flexibility. This process includes discretizing the environment and applying graph search algorithms to find relay chains and relay trees representing the locations where the UAVs should be placed. Graph search algorithms allow a great deal of flexibility, e.g. when choosing what factors to use for evaluating suitability of UAV placement.

We have presented two new algorithms for calculating relay chains for surveillance of a single target. One algorithm is tailored towards calculating a chain using at most a given number of UAVs. The other algorithm calculates a set of Pareto-optimal relay chains, and lets the ground operator choose between the different alternatives.

The problem of simultaneous surveillance of several targets is considerably more difficult since it adds further restrictions to the already NP-hard Steiner tree problem. Very few existing algorithms are applicable to our problem, as we require that all surveillance targets are leaves in the tree. Several of the algorithms that can model this requirement cannot be used due to memory requirements or assumptions about the graph. To quickly generate relay trees for such situations, we modify the cheapest path heuristic. Due to its flexible nature, we can use it to solve problems involving a large variety of restrictions and extensions. Once a relay tree has been calculated, we apply another algorithm to continually improve the tree. The algorithm performs local optimizations of the tree and each such improvement yields an improvement of the complete tree. Once such an improved tree has been found, it can be immediately displayed to the ground operator. The process of improvement can be performed until the algorithm can no

longer improve the tree or until a certain time has passed.

The algorithms have been implemented and integrated into the UAS infrastructure developed at our lab. We have described the implementation as well as the testing that has been performed. The algorithms have been tested in both simulated environments as well as discretizations of real environments.

Although we are able to solve some problems optimally or close to optimally, there are still potential for improvement and expansion in many areas. Some of the areas are discussed in the next section.

8.1 Future Research

A discretization created from a three-dimensional grid was used for testing of the algorithms presented here and several other discretization options were also discussed. It would be interesting to make a structured evaluation of different discretization approaches for different environments to determine which, or more likely which combinations of, discretizations that are most suitable for different environments. For example, in an urban environment, it appears likely that a grid combined with an expanded geometry graph will provide a good compromise between memory requirement and quality of relay chains and trees.

In the **STR-MinCostLimited** problem, an optimization is performed in order to find the least-cost chain given a limit on the number of hops in the chain. This is an example of a bi-objective problem where one objective function can be specified freely while other is fixed. The dual ascent algorithm can be generalized to handle problems where both objectives can be specified freely, something that can be very useful in the context of relays. As an example: when micro-UAVs are used as relays it can be necessary to explore the trade-offs between transmission quality and tolerance to wind drift for different positioning alternatives. The kind of problem suggested here is in its general form the NP-hard constrained shortest path problem [10, 39], which is often solved using Lagrangian relaxation, based on the same dual function as the dual ascent method presented here, but with other dual search techniques.

In this thesis, we assumed that the targets were stationary and that their positions were known. Obvious extensions are lifting one or more of these assumptions, and to provide relay chains to a moving target or relay trees to several moving targets. This increases the complexity of the problem considerably. As an example, it might be impossible to maintain constant communication between the surveillance UAV and the base station as the UAVs have to temporarily move behind buildings and outside the commu-

nication range. For such situations, an approach where it is guaranteed that information can flow from the surveillance UAV to the base station at certain time intervals, e.g. every fifth second, might be sufficient.

Bibliography

[1] AIICS. `http://www.ida.liu.se/divisions/aiics/`. Accessed January 14, 2011.

[2] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] Jamal N. Al-Karaki and Ahmed E. Kamal. Routing techniques in wireless sensor networks: a survey. *IEEE Wireless Communications*, 11(6):6–28, December 2004.

[4] Nancy M. Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1996.

[5] Stuart O. Anderson, Reid Simmons, and Dani Goldberg. Maintaining Line of Sight Communications Network between Planetary rovers. In *Proceedings of the 2003 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*, pages 2266–2272, 2003.

[6] Ronald C. Arkin and Jonathan Diaz. Line-of-sight constrained exploration for reactive multiagent robotic teams. In *7th International Workshop on Advanced Motion Control*, pages 455–461, 2002.

[7] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[8] Anantaram Balakrishnan and Kemal Altinkemer. Using a hop-constrained model to generate alternative communication network design. *ORSA Journal of Computing*, 4:192–205, 1992.

[9] Randal Beard, Derek Kingston, Morgan Quigley, Deryl Snyder, Reed Christiansen, Walt Johnson, Timothy McLain, and Michael A.

Goodrich. Autonomous vehicle technologies for small fixed-wing UAVs. *Journal of Aerospace Computing, Information, and Communication*, 2(1):92–108, January 2005.

[10] John E. Beasley and Nicos Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, July 1989.

[11] Paul Blaer. Robot path planning using generalized Voronoi diagrams. `http://www1.cs.columbia.edu/~pblaer/projects/path_planner/voronoi.html`. Accessed October 13, 2009.

[12] Valérie Boor, Mark H. Overmars, and A. Frank van der Stappen. Gaussian sampling for probabilistic roadmap planners. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2001.

[13] Scott A. Bortoff. Path planning for UAVs. In *Proceedings of the American Control Conference*, pages 364–368, vol.1, 2000.

[14] Timothy X. Brown, Brian Argrow, Cory Dixon, Sheetalkumar Doshi, Roshan-George Thekkekunnel, and Daniel Henkel. Ad hoc UAV ground network (AUGNet). In *Proceedings of the AIAA 3rd "Unmanned Unlimited" Technical Conference*, 2004.

[15] Oleg Burdakov, Patrick Doherty, Kaj Holmberg, Jonas Kvarnström, and Per-Magnus Olsson. Positioning unmanned aerial vehicles as communication relays for surveillance tasks. In *Proceedings of the 2009 Conference on Robotics: Science and Systems (RSS)*, pages 257–264, 2009.

[16] Oleg Burdakov, Patrick Doherty, Kaj Holmberg, Jonas Kvarnström, and Per-Magnus Olsson. Relay positioning for unmanned aerial vehicle surveillance. *International Journal of Robotics Research*, 29(8):1069–1087, 2010.

[17] Oleg Burdakov, Patrick Doherty, Kaj Holmberg, and Per-Magnus Olsson. Optimal placement of UV-based communications relay nodes. *Journal of Global Optimization*, 48(4):511–531, 2010.

[18] Oleg Burdakov, Kaj Holmberg, and Per-Magnus Olsson. A dual ascent method for the hop-constrained shortest path with application to positioning of unmanned air vehicles. Technical Report LiTH-MAT-R-2008-07, Department of Mathematics, Linköping University, 2008.

[19] Oleg Burdakov, Kaj Holmberg, and Per-Magnus Olsson. A dual ascent method for the hop-constrained shortest path with application to positioning of unmanned air vehicles. *Naval Research Logistics*, submitted in 2010.

[20] Carmen Cerasoli. An analysis of unmanned airborne vehicle relay coverage in urban environments. In *Proceedings of MILCOM 2007*. IEEE, 2007.

[21] Phillip R. Chandler, Meir Pachter, and Steven Rasmussen. UAV cooperative control. In *Proceedings of the American Control Conference*, pages 50–55, 2001.

[22] Moses Charikar, Chandra Chekuri, To-yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed Steiner problems. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 192–200, 1998.

[23] Nang-Ping Chen. New algorithms for the Steiner tree on graphs. In *IEEE Symposium on Circuits and Systems*, pages 1217–1219, 1983.

[24] Yu Ming Chen, Liang Dong, and Jun-Seok Oh. Real-time video relay for UAV traffic surveillance systems through available communication networks. In *Wireless Communications and Networking Conference (WCNC)*, 2007.

[25] Chen-Mou, Cheng, Pai-Hsiang Hsiao, H. T. Kung, and Dario Vlahh. Maximizing throughput of UAV-relaying networks with the load-carry-and-deliver paradigm. In *Wireless Communications and Networking Conference (WCNC)*, 2007.

[26] Sunil Chopra and M. R. Rao. The Steiner tree problem ii: Properties and classes of facets. *Mathematical Programming*, 64(1):231–246, 1994.

[27] Gianpaolo Conte, Maria Hempel, Piotr Rudol, David Lundström, Simone Duranti, Mariusz Wzorek, and Patrick Doherty. High accuracy ground target geo-location using autonomous micro aerial vehicle platforms. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2008.

[28] CORBA. `http://www.omg.org/gettingstarted/corbafaq.htm`. Accessed April 19, 2010.

[29] Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[30] Alysson M. Costa, Jean-Francois Cordeau, and Gilbert Laporte. Fast heuristics for the Steiner tree problem with revenues, budget and hop constraints. *European Journal of Operational Research*, 190(1):68–78, 2008.

[31] Geir Dahl and Luis Gouveia. On the directed hop-constrained shortest path problem. *Operations Research Letters*, 32(1):15–22, January 2004.

[32] Edsger Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[33] Patrick Doherty, Patrik Haslum, Fredrik Heintz, Torsten Merz, Per Nyblom, Tommy Persson, and Björn Wingman. A Distributed Architecture for Autonomous Unmanned Aerial Vehicle Experimentation. In *Proceedings of the 7th International Symposium on Distributed Autonomous Systems*, pages 221–230, 2004.

[34] Patrick Doherty, Jonas Kvarnström, Fredrik Heintz, David Landén, and Per-Magnus Olsson. Research with collaborative unmanned aircraft systems. In *Proceedings of the Dagstuhl Workshop on Cognitive Robotics*, 2010.

[35] Patrick Doherty, David Landén, and Fredrik Heintz. A distributed task specification language for mixed-initiative delegation. In *Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA-2010)*, 2010.

[36] Doratha E. Drake and Stefan Hougardy. On approximation algorithms for the terminal Steiner tree problem. *Information Processing Letters*, 89:15–18, 2004.

[37] Ding-Zhu Du, Bing Lu, Hung Ngo, and Panos M. Pardalos. *Steiner tree problems*. Kluwer Academic Publishers, 2001.

[38] Ding-Zhu Du, J. MacGregor Smith, and J.H. Rubenstein. *Advances in Steiner Trees*. Kluwer Academic Publishers, 2000.

[39] Irina Dumitrescu and Natashia Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks*, 42(3):135–153, 2003.

[40] Simone Duranti, Gianpaolo Conte, David Lundström, Piotr Rudol, Mariusz Wzorek, and Patrick Doherty. LinkMAV, a prototype rotary wing micro aerial vehicle. In *Proceedings of the 17th IFAC Symposium on Automatic Control in Aerospace*, 2007.

[41] Joel M. Esposito and Thomas W. Dunbar. Maintaining wireless connectivity constraints for swarms in the presence of obstacles. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 946–951, 2006.

[42] Marcia Fampa and Kurt M. Anstreicher. An improved algorithm for computing Steiner minimal trees in Euclidean d-space. *Discrete Optimization*, 5(2):530–540, 2008.

[43] R. A. Finkell and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[44] David Gesbert, Mansoor Shafi, Da shan Shiu, Peter J. Smith, and Ayman Naguib. From theory to practice: an overview of MIMO space-time coded wireless systems. *IEEE Journal on Selected Areas in Communications*, 21(3):281–302, 2003.

[45] Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*. Springer, 2003.

[46] Roch Guérin and Ariel Orda. Computing shortest paths for any number of hops. *IEEE/ACM Transactions on Networking*, 10(5), 2002.

[47] Zhu Han, A. Lee Swindlehurst, and K. J. Ray Liu. Smart deployment/movement of unmanned air vehicle to improve connectivity in MANET. In *Wireless Communications and Networking Conference (WCNC)*, 2006.

[48] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost graphs. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 1968.

[49] Donald Hearn and M. Pauline Baker. *Computer Graphics with OpenGL. 3rd Edition*. Prentice Hall, 2003.

[50] Ming-I Hsieh, Eric Hsiao-Kuang Wu, and Men-Feng Tsai. FasterDSP: A faster approximation algorithm for directed Steiner tree problem. *Journal of Information Science and Engineering*, 22:1409–1425, 2006.

[51] Sun-Yuan Hsieh and Huang-Ming Gao. On the partial terminal Steiner tree problem. *Journal of Supercomputing*, 41(1):41–52, 2007.

[52] Sun-Yuan Hsieh and Wen-Hao Pi. On the partial-terminal Steiner tree problem. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 173–177, 2008.

[53] David Hsu, Tingting Jiang, John Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2003.

[54] Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner Tree Problem*. North-Holland, 1992.

[55] David B. Johnson and David A. Maltz. *Dynamic Source Routing In Ad Hoc Wireless Networks*. Kluwer Academic Publishers, 1996.

[56] J.R. Ford Jr. Network flow theory. Technical report, The Rand Corporation, 1956. Technical Paper P-923.

[57] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[58] David Landén, Fredrik Heintz, and Patrick Doherty. Complex task allocation in mixed-initiative delegation: A UAV case study (work in progress). In *Proceedings of the 13th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA–2010)*, 2010.

[59] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report Technical Report 98-11, Computer Science Dept., Iowa State University, 1998.

[60] Steven M. LaValle. *Planning Algorithms*. Cambridge Press, 2006.

[61] Eugene L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[62] Qun Li and Daniela Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 44–55, 2000.

[63] Shu Lin and Daniel J. Costello. *Error Control Coding (2nd Edition)*. Prentice Hall, 2004.

[64] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–670, 1979.

[65] Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt. Bicriteria network design problems. *Journal of Algorithms*, 28(1):142–171, 1998.

[66] Fábio Viduani Martinez, José Coelho de Pina, and José Soares. Algorithms for terminal Steiner trees. *Theoretical Compuer Science*, 389:133–142, 2007.

[67] Martin Mauve, Jörg Widmer, and Hannes Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network*, 15(6):30–39, November/December 2001.

[68] Timothy W. McLain and Randal W. Beard. Cooperative rendezvous of multiple unmanned air vehicles. In *AIAA Guidance, Navigation and Control Conference*, 2000.

[69] Kaisa Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[70] Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 15. North Holland, 1999.

[71] Alejandro R. Mosteo and Luis Montano. Concurrent tree traversals for improved mission performance under limited communication range. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[72] Alejandro R. Mosteo, Luis Montano, and Michail G. Lagoudakis. Guaranteed-performance multi-robot routing under limited communication range. In Hajime Asama, Haruhisa Kurokawa, Jun Ota, and Kosuke Sekiyama, editors, *Distributed Autonomous Robotic Systems 8*, pages 491–502. Springer Berlin Heidelberg, 2009.

[73] Hoa G. Nguyen, Narek Pezeshkian, Michelle Raymond, A. Gupta, and Joseph M. Spector. Autonomous communication relays for tactical robots. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, 2003.

[74] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization, Second Edition*. Springer, 2006.

[75] Claude Oestges and Bruno Clerckx. *MIMO Wireless Communications: From Real-World Propagation to Space-Time Code Design*. Academic Press, 2007.

[76] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley, 2000.

[77] Per-Magnus Olsson, Jonas Kvarnström, Patrick Doherty, Oleg Burdakov, and Kaj Holmberg. Generating UAV communication networks for monitoring and surveillance. In *Proceedings of the International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2010.

[78] OpenStreetMap. `http://www.openstreetmap.org`. Accessed January 15, 2010.

[79] Ramesh Palat, Annamalai Annamalai, and Jeffrey Reed. Cooperative relaying for ad-hoc ground networks using swarm UAVs. In *Proceedings of MILCOM 2006*, 2006.

[80] Per Olof Pettersson. Sampling-based path planning for an autonomous helicopter.

[81] Frank J. Pinkney, Dan Hampel, and Stef DiPierro. Unmanned aerial vehicle (UAV) communications relay. In *Proceedings of MILCOM 1996*. IEEE, 1996.

[82] Gabriel Robins and Alexander Zelikovsky. Improved Steiner tree approximation in graphs. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 770–779, 2000.

[83] Martijn N. Rooker and Andreas Birk. Multi robot exploration under the constraints of wireless networking. *Control Engineering Practice*, 15(4):435–445, 2007.

[84] Steven Roos. Scheduling for ReMove and other partially connected architectures. Technical Report 1–68340–44(2001)–05, Laboratory of Information Technology and Systems, Delft University of Technology, 2001.

[85] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, 2006.

[86] Tom Schouwenaars. *Safe Trajectory Planning of Autonomous Vehicles*. PhD thesis, Massachusetts Institute of Technology, February 2006.

[87] Tom Schouwenaars, Andrew Stubbs, James Paduano, and Eric Feron. Multivehicle path planning for nonline-of-sight communication. *Journal of Field Robotics*, 23(3/4):269–290, 2006.

[88] Mirko Vujošević and Milan Stanojević. A bicriterion Steiner tree problem on graph. *Yugoslav Journal of Operations Research*, 13(1):25–33, 2003.

[89] Andrea Simonetto, Paul Scerri, and Katia Sycara. A mobile network for mobile sensors. In *Proceedings of the 11th International Conference on Information Fusion*, 2008.

[90] Vinay Sridhara and Stephan Bohacek. Realistic propagation simulation of urban mesh networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51(12):3392–3412, 2007.

[91] John Sweeney, TJ Brunette, Yuandong Yang, and Roderic Grupen. Coordinated teams of reactive mobile platforms. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[92] Hiromitsu Takahashi and Akira Matsuyama. An approximate solution for the Steiner problem in graphs. *Mathematica Japonica*, 24, 1980.

[93] UAS Technologies Sweden AB. `www.uastech.com`.

[94] Martin Thimm. On the approximability of the Steiner tree problem. In J. Sgall, A. Pultr, and P. Kolman, editors, *Lecture Notes in Computer Science 2136: Mathematical Foundations of Computer Science 2001*, pages 678–689. Springer Berlin / Heidelberg, 2001.

[95] David Tse and Pramod Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[96] Stefan Voß. Worst-case performance of some heuristics for Steiner's problem in directed graphs. *Information Processing Letters*, 48:99–105, 1993.

[97] Stefan Voß. The Steiner tree problem with hop constraints. *Annals of Operations Research*, 86(0):321–345, 1999.

[98] Stefan Voß. Modern heuristic search methods for the Steiner tree problem in graphs. In Ding-Zhu Du, J. MacGregor Smith, and J.H. Rubenstein, editors, *Advances in Steiner Trees*, pages 283–323. Kluwer Academic Publishers, 2000.

[99] Jean-Frédéric Wagen and Karim Rizk. Radiowave propagation, building databases, and GIS: anything in common? A radio engineer's viewpoint. *Environment and Planning B: Planning and Design*, 30(5):767–787, September 2003.

[100] Pawel Winter. Steiner problems in networks: A survey. *Networks*, pages 129–167, 1987.

[101] Martin Zachariasen and Pawel Winter. Obstacle-avoiding euclidean Steiner trees in the plane: An exact algorithm. In M.T. Goodrich and C.C. McGeoch, editors, *Lecture Notes in Computer Science 1619: Algorithm Engineering and Experimentation*, pages 286–299. Springer, 1999.

[102] Pengcheng Zhan. *Optimizing Wireless Throughput: Methods and Applications*. PhD thesis, Brigham Young University, 2007.

[103] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

[104] Leonid Zosin and Samir Khuller. On directed Steiner trees. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 59–63, 2002.

Avdelning, Institution
Division, Department

AIICS,
Dept. of Computer and Information Science
581 83 Linköping

Datum
Date

April 20, 2011

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ _____

Rapporttyp
Report category

☒ Licentiatavhandling
☐ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ _____

ISBN
978-91-7393-200-4

ISRN
LiU-Tek-Lic–2011:15

Serietitel och serienummer **ISSN**
Title of series, numbering
0280–7971

Linköping Studies in Science and Technology

Thesis No. 1476

URL för elektronisk version
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-66060

Titel
Title

Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles

Författare
Author

Per-Magnus Olsson

Sammanfattning
Abstract

Surveillance is an important application for unmanned aerial vehicles (UAVs). The sensed information often has high priority and it must be made available to human operators as quickly as possible. Due to obstacles and limited communication range, it is not always possible to transmit the information directly to the base station. In this case, other UAVs can form a relay chain between the surveillance UAV and the base station. Determining suitable positions for such UAVs is a complex optimization problem in and of itself, and is made even more difficult by communication and surveillance constraints.

To solve different variations of finding positions for UAVs for surveillance of one target, two new algorithms have been developed. One of the algorithms is developed especially for finding a set of relay chains offering different trade-offs between the number of UAVs and the quality of the chain. The other algorithm is tailored towards finding the highest quality chain possible, given a limited number of available UAVs.

Finding the optimal positions for surveillance of several targets is more difficult. A study has been performed, in order to determine how the problems of interest can be solved. It turns out that very few of the existing algorithms can be used due to the characteristics of our specific problem. For this reason, an algorithm for quickly calculating positions for surveillance of multiple targets has been developed. This enables calculation of an initial chain that is immediately made available to the user, and the chain is then incrementally optimized according to the user's desire.

Nyckelord
Keywords

Unmanned aerial vehicles, surveillance, communication relay, label-correcting, dual ascent, Steiner trees

Department of Computer and Information Science
Linköpings universitet

Linköpings Studies in Science and Technology
Faculty of Arts and Sciences – Licentiate Theses

No 17 **Vojin Plavsic:** Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28 **Arne Jönsson, Mikael Patel:** An Interactive Flowcharting Technique for Communicating and Realizing Algorithms, 1984.

No 29 **Johnny Eckerland:** Retargeting of an Incremental Code Generator, 1984.

No 48 **Henrik Nordin:** On the Use of Typical Cases for Knowledge-Based Consultation and Teaching, 1985.

No 52 **Zebo Peng:** Steps Towards the Formalization of Designing VLSI Systems, 1985.

No 60 **Johan Fagerström:** Simulation and Evaluation of Architecture based on Asynchronous Processes, 1985.

No 71 **Jalal Maleki:** ICONStraint, A Dependency Directed Constraint Maintenance System, 1987.

No 72 **Tony Larsson:** On the Specification and Verification of VLSI Systems, 1986.

No 73 **Ola Strömfors:** A Structure Editor for Documents and Programs, 1986.

No 74 **Christos Levcopoulos:** New Results about the Approximation Behavior of the Greedy Triangulation, 1986.

No 104 **Shamsul I. Chowdhury: S**tatistical Expert Systems - a Special Application Area for Knowledge-Based Computer Methodology, 1987.

No 108 **Rober Bilos:** Incremental Scanning and Token-Based Editing, 1987.

No 111 **Hans Block:** SPORT-SORT Sorting Algorithms and Sport Tournaments, 1987.

No 113 **Ralph Rönnquist:** Network and Lattice Based Approaches to the Representation of Knowledge, 1987.

No 118 **Mariam Kamkar, Nahid Shahmehri:** Affect-Chaining in Program Flow Analysis Applied to Queries of Programs, 1987.

No 126 **Dan Strömberg:** Transfer and Distribution of Application Programs, 1987.

No 127 **Kristian Sandahl:** Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems, 1987.

No 139 **Christer Bäckström:** Reasoning about Interdependent Actions, 1988.

No 140 **Mats Wirén:** On Control Strategies and Incrementality in Unification-Based Chart Parsing, 1988.

No 146 **Johan Hultman:** A Software System for Defining and Controlling Actions in a Mechanical System, 1988.

No 150 **Tim Hansen:** Diagnosing Faults using Knowledge about Malfunctioning Behavior, 1988.

No 165 **Jonas Löwgren:** Supporting Design and Management of Expert System User Interfaces, 1989.

No 166 **Ola Petersson:** On Adaptive Sorting in Sequential and Parallel Models, 1989.

No 174 **Yngve Larsson:** Dynamic Configuration in a Distributed Environment, 1989.

No 177 **Peter Åberg:** Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181 **Henrik Eriksson:** A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184 **Ivan Rankin:** The Deep Generation of Text in Expert Critiquing Systems, 1989.

No 187 **Simin Nadjm-Tehrani:** Contributions to the Declarative Approach to Debugging Prolog Programs, 1989.

No 189 **Magnus Merkel:** Temporal Information in Natural Language, 1989.

No 196 **Ulf Nilsson:** A Systematic Approach to Abstract Interpretation of Logic Programs, 1989.

No 197 **Staffan Bonnier:** Horn Clause Logic with External Procedures: Towards a Theoretical Framework, 1989.

No 203 **Christer Hansson:** A Prototype System for Logical Reasoning about Time and Action, 1990.

No 212 **Björn Fjellborg:** An Approach to Extraction of Pipeline Structures for VLSI High-Level Synthesis, 1990.

No 230 **Patrick Doherty:** A Three-Valued Approach to Non-Monotonic Reasoning, 1990.

No 237 **Tomas Sokolnicki:** Coaching Partial Plans: An Approach to Knowledge-Based Tutoring, 1990.

No 250 **Lars Strömberg:** Postmortem Debugging of Distributed Systems, 1990.

No 253 **Torbjörn Näslund:** SLDFA-Resolution - Computing Answers for Negative Queries, 1990.

No 260 **Peter D. Holmes:** Using Connectivity Graphs to Support Map-Related Reasoning, 1991.

No 283 **Olof Johansson:** Improving Implementation of Graphical User Interfaces for Object-Oriented Knowledge- Bases, 1991.

No 298 **Rolf G Larsson:** Aktivitetsbaserad kalkylering i ett nytt ekonomisystem, 1991.

No 318 **Lena Strömbäck:** Studies in Extended Unification-Based Formalism for Linguistic Description: An Algorithm for Feature Structures with Disjunction and a Proposal for Flexible Systems, 1992.

No 319 **Mikael Pettersson:** DML-A Language and System for the Generation of Efficient Compilers from Denotational Specification, 1992.

No 326 **Andreas Kågedal:** Logic Programming with External Procedures: an Implementation, 1992.

No 328 **Patrick Lambrix:** Aspects of Version Management of Composite Objects, 1992.

No 333 **Xinli Gu:** Testability Analysis and Improvement in High-Level Synthesis Systems, 1992.

No 335 **Torbjörn Näslund:** On the Role of Evaluations in Iterative Development of Managerial Support Systems, 1992.

No 348 **Ulf Cederling:** Industrial Software Development - a Case Study, 1992.

No 352 **Magnus Morin:** Predictable Cyclic Computations in Autonomous Systems: A Computational Model and Implementation, 1992.

No 371 **Mehran Noghabai:** Evaluation of Strategic Investments in Information Technology, 1993.

No 378 **Mats Larsson:** A Transformational Approach to Formal Digital System Design, 1993.

No 380 **Johan Ringström:** Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381 **Michael Jansson:** Propagation of Change in an Intelligent Information System, 1993.
No 383 **Jonni Harrius:** An Architecture and a Knowledge Representation Model for Expert Critiquing Systems, 1993.
No 386 **Per Österling:** Symbolic Modelling of the Dynamic Environments of Autonomous Agents, 1993.
No 398 **Johan Boye:** Dependency-based Groudness Analysis of Functional Logic Programs, 1993.
No 402 **Lars Degerstedt:** Tabulated Resolution for Well Founded Semantics, 1993.
No 406 **Anna Moberg:** Satellitkontor - en studie av kommunikationsmönster vid arbete på distans, 1993.
No 414 **Peter Carlsson:** Separation av företagsledning och finansiering - fallstudier av företagsledarutköp ur ett agent-teoretiskt perspektiv, 1994.
No 417 **Camilla Sjöström:** Revision och lagreglering - ett historiskt perspektiv, 1994.
No 436 **Cecilia Sjöberg:** Voices in Design: Argumentation in Participatory Development, 1994.
No 437 **Lars Viklund:** Contributions to a High-level Programming Environment for a Scientific Computing, 1994.
No 440 **Peter Loborg:** Error Recovery Support in Manufacturing Control Systems, 1994.
FHS 3/94 **Owen Eriksson:** Informationssystem med verksamhetskvalitet - utvärdering baserat på ett verksamhetsinriktat och samskapande perspektiv, 1994.
FHS 4/94 **Karin Pettersson:** Informationssystemstrukturering, ansvarsfördelning och användarinflytande - En komparativ studie med utgångspunkt i två informationssystemstrategier, 1994.
No 441 **Lars Poignant:** Informationsteknologi och företagsetablering - Effekter på produktivitet och region, 1994.
No 446 **Gustav Fahl:** Object Views of Relational Data in Multidatabase Systems, 1994.
No 450 **Henrik Nilsson:** A Declarative Approach to Debugging for Lazy Functional Languages, 1994.
No 451 **Jonas Lind:** Creditor - Firm Relations: an Interdisciplinary Analysis, 1994.
No 452 **Martin Sköld:** Active Rules based on Object Relational Queries - Efficient Change Monitoring Techniques, 1994.
No 455 **Pär Carlshamre:** A Collaborative Approach to Usability Engineering: Technical Communicators and System Developers in Usability-Oriented Systems Development, 1994.
FHS 5/94 **Stefan Cronholm:** Varför CASE-verktyg i systemutveckling? - En motiv- och konsekvensstudie avseende arbetssätt och arbetsformer, 1994.
No 462 **Mikael Lindvall:** A Study of Traceability in Object-Oriented Systems Development, 1994.
No 463 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av Sandviks förvärv av Bahco Verktyg, 1994.
No 464 **Hans Olsén:** Collage Induction: Proving Properties of Logic Programs by Program Synthesis, 1994.
No 469 **Lars Karlsson:** Specification and Synthesis of Plans Using the Features and Fluents Framework, 1995.
No 473 **Ulf Söderman:** On Conceptual Modelling of Mode Switching Systems, 1995.
No 475 **Choong-ho Yi:** Reasoning about Concurrent Actions in the Trajectory Semantics, 1995.
No 476 **Bo Lagerström:** Successiv resultatavräkning av pågående arbeten. - Fallstudier i tre byggföretag, 1995.
No 478 **Peter Jonsson:** Complexity of State-Variable Planning under Structural Restrictions, 1995.
FHS 7/95 **Anders Avdic:** Arbetsintegrerad systemutveckling med kalkylprogram, 1995.
No 482 **Eva L Ragnemalm:** Towards Student Modelling through Collaborative Dialogue with a Learning Companion, 1995.
No 488 **Eva Toller:** Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
No 489 **Erik Stoy:** A Petri Net Based Unified Representation for Hardware/Software Co-Design, 1995.
No 497 **Johan Herber:** Environment Support for Building Structured Mathematical Models, 1995.
No 498 **Stefan Svenberg:** Structure-Driven Derivation of Inter-Lingual Functor-Argument Trees for Multi-Lingual Generation, 1995.
No 503 **Hee-Cheol Kim:** Prediction and Postdiction under Uncertainty, 1995.
FHS 8/95 **Dan Fristedt:** Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
FHS 9/95 **Malin Bergvall:** Systemförvaltning i praktiken - en kvalitativ studie avseende centrala begrepp, aktiviteter och ansvarsroller, 1995.
No 513 **Joachim Karlsson:** Towards a Strategy for Software Requirements Selection, 1995.
No 517 **Jakob Axelsson:** Schedulability-Driven Partitioning of Heterogeneous Real-Time Systems, 1995.
No 518 **Göran Forslund:** Toward Cooperative Advice-Giving Systems: The Expert Systems Experience, 1995.
No 522 **Jörgen Andersson:** Bilder av småföretagares ekonomistyrning, 1995.
No 538 **Staffan Flodin:** Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 545 **Vadim Engelson:** An Approach to Automatic Construction of Graphical User Interfaces for Applications in Scientific Computing, 1996.
No 546 **Magnus Werner :** Multidatabase Integration using Polymorphic Queries and Views, 1996.
FiF-a 1/96 **Mikael Lind:** Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 549 **Jonas Hallberg:** High-Level Synthesis under Local Timing Constraints, 1996.
No 550 **Kristina Larsen:** Förutsättningar och begränsningar för arbete på distans - erfarenheter från fyra svenska företag. 1996.
No 557 **Mikael Johansson:** Quality Functions for Requirements Engineering Methods, 1996.
No 558 **Patrik Nordling:** The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 561 **Anders Ekman:** Exploration of Polygonal Environments, 1996.
No 563 **Niclas Andersson:** Compilation of Mathematical Models to Parallel Code, 1996.
No 567 **Johan Jenvald:** Simulation and Data Collection in Battle Training, 1996.

No 575 **Niclas Ohlsson:** Software Quality Engineering by Early Identification of Fault-Prone Modules, 1996.

No 576 **Mikael Ericsson:** Commenting Systems as Design Support—A Wizard-of-Oz Study, 1996.

No 587 **Jörgen Lindström:** Chefers användning av kommunikationsteknik, 1996.

No 589 **Esa Falkenroth:** Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.

No 591 **Niclas Wahllöf:** A Default Extension to Description Logics and its Applications, 1996.

No 595 **Annika Larsson:** Ekonomisk Styrning och Organisatorisk Passion - ett interaktivt perspektiv, 1997.

No 597 **Ling Lin:** A Value-based Indexing Technique for Time Sequences, 1997.

No 598 **Rego Granlund:** C^3Fire - A Microworld Supporting Emergency Management Training, 1997.

No 599 **Peter Ingels:** A Robust Text Processing Technique Applied to Lexical Error Recovery, 1997.

No 607 **Per-Arne Persson:** Toward a Grounded Theory for Support of Command and Control in Military Coalitions, 1997.

No 609 **Jonas S Karlsson:** A Scalable Data Structure for a Parallel Data Server, 1997.

FiF-a 4 **Carita Åbom:** Videomötesteknik i olika affärssituationer - möjligheter och hinder, 1997.

FiF-a 6 **Tommy Wedlund**: Att skapa en företagsanpassad systemutvecklingsmodell - genom rekonstruktion, värdering och vidareutveckling i T50-bolag inom ABB, 1997.

No 615 **Silvia Coradeschi**: A Decision-Mechanism for Reactive and Coordinated Agents, 1997.

No 623 **Jan Ollinen:** Det flexibla kontorets utveckling på Digital - Ett stöd för multiflex? 1997.

No 626 **David Byers:** Towards Estimating Software Testability Using Static Analysis, 1997.

No 627 **Fredrik Eklund:** Declarative Error Diagnosis of GAPLog Programs, 1997.

No 629 **Gunilla Ivefors:** Krigsspel och Informationsteknik inför en oförutsägbar framtid, 1997**.**

No 631 **Jens-Olof Lindh:** Analysing Traffic Safety from a Case-Based Reasoning Perspective, 1997

No 639 **Jukka Mäki-Turja:**. Smalltalk - a suitable Real-Time Language, 1997.

No 640 **Juha Takkinen**: CAFE: Towards a Conceptual Model for Information Management in Electronic Mail, 1997.

No 643 **Man Lin**: Formal Analysis of Reactive Rule-based Programs, 1997.

No 653 **Mats Gustafsson**: Bringing Role-Based Access Control to Distributed Systems, 1997.

FiF-a 13 **Boris Karlsson:** Metodanalys för förståelse och utveckling av systemutvecklingsverksamhet. Analys och värdering av systemutvecklingsmodeller och dess användning, 1997.

No 674 **Marcus Bjäreland:** Two Aspects of Automating Logics of Action and Change - Regression and Tractability, 1998.

No 676 **Jan Håkegård**: Hierarchical Test Architecture and Board-Level Test Controller Synthesis, 1998.

No 668 **Per-Ove Zetterlund**: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.

No 675 **Jimmy Tjäder**: Projektledaren & planen - en studie av projektledning i tre installations- och systemutvecklingsprojekt, 1998.

FiF-a 14 **Ulf Melin**: Informationssystem vid ökad affärs- och processorientering - egenskaper, strategier och utveckling, 1998.

No 695 **Tim Heyer**: COMPASS: Introduction of Formal Methods in Code Development and Inspection, 1998.

No 700 **Patrik Hägglund:** Programming Languages for Computer Algebra, 1998.

FiF-a 16 **Marie-Therese Christiansson:** Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.

No 712 **Christina Wennestam:** Information om immateriella resurser. Investeringar i forskning och utveckling samt i personal inom skogsindustrin, 1998.

No 719 **Joakim Gustafsson:** Extending Temporal Action Logic for Ramification and Concurrency, 1998.

No 723 **Henrik André-Jönsson:** Indexing time-series data using text indexing methods, 1999.

No 725 **Erik Larsson:** High-Level Testability Analysis and Enhancement Techniques, 1998.

No 730 **Carl-Johan Westin:** Informationsförsörjning: en fråga om ansvar - aktiviteter och uppdrag i fem stora svenska organisationers operativa informationsförsörjning, 1998.

No 731 **Åse Jansson:** Miljöhänsyn - en del i företags styrning, 1998.

No 733 **Thomas Padron-McCarthy:** Performance-Polymorphic Declarative Queries, 1998.

No 734 **Anders Bäckström:** Värdeskapande kreditgivning - Kreditriskhantering ur ett agentteoretiskt perspektiv, 1998.

FiF-a 21 **Ulf Seigerroth:** Integration av förändringsmetoder - en modell för välgrundad metodintegration, 1999.

FiF-a 22 **Fredrik Öberg:** Object-Oriented Frameworks - A New Strategy for Case Tool Development, 1998.

No 737 **Jonas Mellin:** Predictable Event Monitoring, 1998.

No 738 **Joakim Eriksson:** Specifying and Managing Rules in an Active Real-Time Database System, 1998.

FiF-a 25 **Bengt E W Andersson:** Samverkande informationssystem mellan aktörer i offentliga åtaganden - En teori om aktörsarenor i samverkan om utbyte av information, 1998.

No 742 **Pawel Pietrzak:** Static Incorrectness Diagnosis of CLP (FD), 1999.

No 748 **Tobias Ritzau:** Real-Time Reference Counting in RT-Java, 1999.

No 751 **Anders Ferntoft:** Elektronisk affärskommunikation - kontaktkostnader och kontaktprocesser mellan kunder och leverantörer på producentmarknader, 1999.

No 752 **Jo Skåmedal:** Arbete på distans och arbetsformens påverkan på resor och resmönster, 1999.

No 753 **Johan Alvehus:** Mötets metaforer. En studie av berättelser om möten, 1999.

No 754 **Magnus Lindahl:** Bankens villkor i låneavtal vid kreditgivning till högt belånade företagsförvärv: En studie ur ett agentteoretiskt perspektiv, 2000.

No 766 **Martin V. Howard:** Designing dynamic visualizations of temporal data, 1999.
No 769 **Jesper Andersson:** Towards Reactive Software Architectures, 1999.
No 775 **Anders Henriksson:** Unique kernel diagnosis, 1999.
FiF-a 30 **Pär J. Ågerfalk:** Pragmatization of Information Systems - A Theoretical and Methodological Outline, 1999.
No 787 **Charlotte Björkegren:** Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 788 **Håkan Nilsson:** Informationsteknik som drivkraft i granskningsprocessen - En studie av fyra revisionsbyråer, 2000.
No 790 **Erik Berglund:** Use-Oriented Documentation in Software Development, 1999.
No 791 **Klas Gäre:** Verksamhetsförändringar i samband med IS-införande, 1999.
No 800 **Anders Subotic:** Software Quality Inspection, 1999.
No 807 **Svein Bergum**: Managerial communication in telework, 2000.
No 809 **Flavius Gruian:** Energy-Aware Design of Digital Systems, 2000.
FiF-a 32 **Karin Hedström:** Kunskapsanvändning och kunskapsutveckling hos verksamhetskonsulter - Erfarenheter från ett FOU-samarbete, 2000.
No 808 **Linda Askenäs:** Affärssystemet - En studie om teknikens aktiva och passiva roll i en organisation, 2000.
No 820 **Jean Paul Meynard:** Control of industrial robots through high-level task programming, 2000.
No 823 **Lars Hult:** Publika Gränsytor - ett designexempel, 2000.
No 832 **Paul Pop:** Scheduling and Communication Synthesis for Distributed Real-Time Systems, 2000.
FiF-a 34 **Göran Hultgren:** Nätverksinriktad Förändringsanalys - perspektiv och metoder som stöd för förståelse och utveckling av affärsrelationer och informationssystem, 2000.
No 842 **Magnus Kald:** The role of management control systems in strategic business units, 2000.
No 844 **Mikael Cäker:** Vad kostar kunden? Modeller för intern redovisning, 2000.
FiF-a 37 **Ewa Braf**: Organisationers kunskapsverksamheter - en kritisk studie av ”knowledge management”, 2000.
FiF-a 40 **Henrik Lindberg:** Webbaserade affärsprocesser - Möjligheter och begränsningar, 2000.
FiF-a 41 **Benneth Christiansson:** Att komponentbasera informationssystem - Vad säger teori och praktik?, 2000.
No. 854 **Ola Pettersson:** Deliberation in a Mobile Robot, 2000.
No 863 **Dan Lawesson**: Towards Behavioral Model Fault Isolation for Object Oriented Control Systems, 2000.
No 881 **Johan Moe:** Execution Tracing of Large Distributed Systems, 2001.
No 882 **Yuxiao Zhao:** XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement, 2001.
No 890 **Annika Flycht-Eriksson:** Domain Knowledge Management in Information-providing Dialogue systems, 2001.
FiF-a 47 **Per-Arne Segerkvis**t: Webbaserade imaginära organisationers samverkansformer: Informationssystemarkitektur och aktörssamverkan som förutsättningar för affärsprocesser, 2001.
No 894 **Stefan Svarén:** Styrning av investeringar i divisionaliserade företag - Ett koncernperspektiv, 2001.
No 906 **Lin Han**: Secure and Scalable E-Service Software Delivery, 2001.
No 917 **Emma Hansson:** Optionsprogram för anställda - en studie av svenska börsföretag, 2001.
No 916 **Susanne Odar:** IT som stöd för strategiska beslut, en studie av datorimplementerade modeller av verksamhet som stöd för beslut om anskaffning av JAS 1982, 2002.
FiF-a-49 **Stefan Holgersson:** IT-system och filtrering av verksamhetskunskap - kvalitetsproblem vid analyser och beslutsfattande som bygger på uppgifter hämtade från polisens IT-system, 2001.
FiF-a-51 **Per Oscarsson:** Informationssäkerhet i verksamheter - begrepp och modeller som stöd för förståelse av informationssäkerhet och dess hantering, 2001.
No 919 **Luis Alejandro Cortes:** A Petri Net Based Modeling and Verification Technique for Real-Time Embedded Systems, 2001.
No 915 **Niklas Sandell:** Redovisning i skuggan av en bankkris - Värdering av fastigheter. 2001.
No 931 **Fredrik Elg:** Ett dynamiskt perspektiv på individuella skillnader av heuristisk kompetens, intelligens, mentala modeller, mål och konfidens i kontroll av mikrovärlden Moro, 2002.
No 933 **Peter Aronsson:** Automatic Parallelization of Simulation Code from Equation Based Simulation Languages, 2002.
No 938 **Bourhane Kadmiry**: Fuzzy Control of Unmanned Helicopter, 2002.
No 942 **Patrik Haslum**: Prediction as a Knowledge Representation Problem: A Case Study in Model Design, 2002.
No 956 **Robert Sevenius:** On the instruments of governance - A law & economics study of capital instruments in limited liability companies, 2002.
FiF-a 58 **Johan Petersson:** Lokala elektroniska marknadsplatser - informationssystem för platsbundna affärer, 2002.
No 964 **Peter Bunus:** Debugging and Structural Analysis of Declarative Equation-Based Languages, 2002.
No 973 **Gert Jervan:** High-Level Test Generation and Built-In Self-Test Techniques for Digital Systems, 2002.
No 958 **Fredrika Berglund:** Management Control and Strategy - a Case Study of Pharmaceutical Drug Development, 2002.
FiF-a 61 **Fredrik Karlsson:** Meta-Method for Method Configuration - A Rational Unified Process Case, 2002.
No 985 **Sorin Manolache:** Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times, 2002.
No 982 **Diana Szentiványi:** Performance and Availability Trade-offs in Fault-Tolerant Middleware, 2002.
No 989 **Iakov Nakhimovski:** Modeling and Simulation of Contacting Flexible Bodies in Multibody Systems, 2002.
No 990 **Levon Saldamli:** PDEModelica - Towards a High-Level Language for Modeling with Partial Differential Equations, 2002.
No 991 **Almut Herzog:** Secure Execution Environment for Java Electronic Services, 2002.
No 999 **Jon Edvardsson:** Contributions to Program- and Specification-based Test Data Generation, 2002.
No 1000 **Anders Arpteg:** Adaptive Semi-structured Information Extraction, 2002.

No 1001 **Andrzej Bednarski:** A Dynamic Programming Approach to Optimal Retargetable Code Generation for Irregular Architectures, 2002.
No 988 **Mattias Arvola:** Good to use! : Use quality of multi-user applications in the home, 2003.
FiF-a 62 **Lennart Ljung:** Utveckling av en projektivitetsmodell - om organisationers förmåga att tillämpa projektarbetsformen, 2003.
No 1003 **Pernilla Qvarfordt:** User experience of spoken feedback in multimodal interaction, 2003.
No 1005 **Alexander Siemers:** Visualization of Dynamic Multibody Simulation With Special Reference to Contacts, 2003.
No 1008 **Jens Gustavsson:** Towards Unanticipated Runtime Software Evolution, 2003.
No 1010 **Calin Curescu:** Adaptive QoS-aware Resource Allocation for Wireless Networks, 2003.
No 1015 **Anna Andersson:** Management Information Systems in Process-oriented Healthcare Organisations, 2003.
No 1018 **Björn Johansson:** Feedforward Control in Dynamic Situations, 2003.
No 1022 **Traian Pop:** Scheduling and Optimisation of Heterogeneous Time/Event-Triggered Distributed Embedded Systems, 2003.
FiF-a 65 **Britt-Marie Johansson:** Kundkommunikation på distans - en studie om kommunikationsmediets betydelse i affärstransaktioner, 2003.
No 1024 **Aleksandra Tešanovic:** Towards Aspectual Component-Based Real-Time System Development, 2003.
No 1034 **Arja Vainio-Larsson**: Designing for Use in a Future Context - Five Case Studies in Retrospect, 2003.
No 1033 **Peter Nilsson:** Svenska bankers redovisningsval vid reservering för befarade kreditförluster - En studie vid införandet av nya redovisningsregler, 2003.
FiF-a 69 **Fredrik Ericsson:** Information Technology for Learning and Acquiring of Work Knowledge, 2003.
No 1049 **Marcus Comstedt:** Towards Fine-Grained Binary Composition through Link Time Weaving, 2003.
No 1052 **Åsa Hedenskog:** Increasing the Automation of Radio Network Control, 2003.
No 1054 **Claudiu Duma:** Security and Efficiency Tradeoffs in Multicast Group Key Management, 2003.
FiF-a 71 **Emma Eliason:** Effektanalys av IT-systems handlingsutrymme, 2003.
No 1055 **Carl Cederberg:** Experiments in Indirect Fault Injection with Open Source and Industrial Software, 2003.
No 1058 **Daniel Karlsson:** Towards Formal Verification in a Component-based Reuse Methodology, 2003.
FiF-a 73 **Anders Hjalmarsson:** Att etablera och vidmakthålla förbättringsverksamhet - behovet av koordination och interaktion vid förändring av systemutvecklingsverksamheter, 2004.
No 1079 **Pontus Johansson:** Design and Development of Recommender Dialogue Systems, 2004.
No 1084 **Charlotte Stoltz:** Calling for Call Centres - A Study of Call Centre Locations in a Swedish Rural Region, 2004.
FiF-a 74 **Björn Johansson:** Deciding on Using Application Service Provision in SMEs, 2004.
No 1094 **Genevieve Gorrell:** Language Modelling and Error Handling in Spoken Dialogue Systems, 2004.
No 1095 **Ulf Johansson:** Rule Extraction - the Key to Accurate and Comprehensible Data Mining Models, 2004.
No 1099 **Sonia Sangari:** Computational Models of Some Communicative Head Movements, 2004.
No 1110 **Hans Nässla:** Intra-Family Information Flow and Prospects for Communication Systems, 2004.
No 1116 **Henrik Sällberg:** On the value of customer loyalty programs - A study of point programs and switching costs, 2004.
FiF-a 77 **Ulf Larsson:** Designarbete i dialog - karaktärisering av interaktionen mellan användare och utvecklare i en systemutvecklingsprocess, 2004.
No 1126 **Andreas Borg:** Contribution to Management and Validation of Non-Functional Requirements, 2004.
No 1127 **Per-Ola Kristensson:** Large Vocabulary Shorthand Writing on Stylus Keyboard, 2004.
No 1132 **Pär-Anders Albinsson:** Interacting with Command and Control Systems: Tools for Operators and Designers, 2004.
No 1130 **Ioan Chisalita:** Safety-Oriented Communication in Mobile Networks for Vehicles, 2004.
No 1138 **Thomas Gustafsson:** Maintaining Data Consistency in Embedded Databases for Vehicular Systems, 2004.
No 1149 **Vaida Jakoniené:** A Study in Integrating Multiple Biological Data Sources, 2005.
No 1156 **Abdil Rashid Mohamed**: High-Level Techniques for Built-In Self-Test Resources Optimization, 2005.
No 1162 **Adrian Pop:** Contributions to Meta-Modeling Tools and Methods, 2005.
No 1165 **Fidel Vascós Palacios:** On the information exchange between physicians and social insurance officers in the sick leave process: an Activity Theoretical perspective, 2005.
FiF-a 84 **Jenny Lagsten:** Verksamhetsutvecklande utvärdering i informationssystemprojekt, 2005.
No 1166 **Emma Larsdotter Nilsson:** Modeling, Simulation, and Visualization of Metabolic Pathways Using Modelica, 2005.
No 1167 **Christina Keller:** Virtual Learning Environments in higher education. A study of students' acceptance of educational technology, 2005.
No 1168 **Cécile Åberg:** Integration of organizational workflows and the Semantic Web, 2005.
FiF-a 85 **Anders Forsman:** Standardisering som grund för informationssamverkan och IT-tjänster - En fallstudie baserad på trafikinformationstjänsten RDS-TMC, 2005.
No 1171 **Yu-Hsing Huang:** A systemic traffic accident model, 2005.
FiF-a 86 **Jan Olausson:** Att modellera uppdrag - grunder för förståelse av processinriktade informationssystem i transaktionsintensiva verksamheter, 2005.
No 1172 **Petter Ahlström**: Affärsstrategier för seniorbostadsmarknaden, 2005.
No 1183 **Mathias Cöster**: Beyond IT and Productivity - How Digitization Transformed the Graphic Industry, 2005.
No 1184 **Åsa Horzella**: Beyond IT and Productivity - Effects of Digitized Information Flows in Grocery Distribution, 2005.
No 1185 **Maria Kollberg**: Beyond IT and Productivity - Effects of Digitized Information Flows in the Logging Industry, 2005.
No 1190 **David Dinka**: Role and Identity - Experience of technology in professional settings, 2005.
No 1191 **Andreas Hansson**: Increasing the Storage Capacity of Recursive Auto-associative Memory by Segmenting Data, 2005.

No 1192 **Nicklas Bergfeldt:** Towards Detached Communication for Robot Cooperation, 2005.
No 1194 **Dennis Maciuszek:** Towards Dependable Virtual Companions for Later Life, 2005.
No 1204 **Beatrice Alenljung:** Decision-making in the Requirements Engineering Process: A Human-centered Approach, 2005.
No 1206 **Anders Larsson:** System-on-Chip Test Scheduling and Test Infrastructure Design, 2005.
No 1207 **John Wilander:** Policy and Implementation Assurance for Software Security, 2005.
No 1209 **Andreas Käll:** Översättningar av en managementmodell - En studie av införandet av Balanced Scorecard i ett landsting, 2005.
No 1225 **He Tan:** Aligning and Merging Biomedical Ontologies, 2006.
No 1228 **Artur Wilk:** Descriptive Types for XML Query Language Xcerpt, 2006.
No 1229 **Per Olof Pettersson:** Sampling-based Path Planning for an Autonomous Helicopter, 2006.
No 1231 **Kalle Burbeck:** Adaptive Real-time Anomaly Detection for Safeguarding Critical Networks, 2006.
No 1233 **Daniela Mihailescu:** Implementation Methodology in Action: A Study of an Enterprise Systems Implementation Methodology, 2006.
No 1244 **Jörgen Skågeby:** Public and Non-public gifting on the Internet, 2006.
No 1248 **Karolina Eliasson:** The Use of Case-Based Reasoning in a Human-Robot Dialog System, 2006.
No 1263 **Misook Park-Westman:** Managing Competence Development Programs in a Cross-Cultural Organisation - What are the Barriers and Enablers, 2006.
FiF-a 90 **Amra Halilovic:** Ett praktikperspektiv på hantering av mjukvarukomponenter, 2006.
No 1272 **Raquel Flodström**: A Framework for the Strategic Management of Information Technology, 2006.
No 1277 **Viacheslav Izosimov:** Scheduling and Optimization of Fault-Tolerant Embedded Systems, 2006.
No 1283 **Håkan Hasewinkel:** A Blueprint for Using Commercial Games off the Shelf in Defence Training, Education and Research Simulations, 2006.
FiF-a 91 **Hanna Broberg:** Verksamhetsanpassade IT-stöd - Designteori och metod, 2006.
No 1286 **Robert Kaminski:** Towards an XML Document Restructuring Framework, 2006.
No 1293 **Jiri Trnka:** Prerequisites for data sharing in emergency management, 2007.
No 1302 **Björn Hägglund:** A Framework for Designing Constraint Stores, 2007.
No 1303 **Daniel Andreasson**: Slack-Time Aware Dynamic Routing Schemes for On-Chip Networks, 2007.
No 1305 **Magnus Ingmarsson:** Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing, 2007.
No 1306 **Gustaf Svedjemo**: Ontology as Conceptual Schema when Modelling Historical Maps for Database Storage, 2007.
No 1307 **Gianpaolo Conte**: Navigation Functionalities for an Autonomous UAV Helicopter, 2007.
No 1309 **Ola Leifler**: User-Centric Critiquing in Command and Control: The DKExpert and ComPlan Approaches, 2007.
No 1312 **Henrik Svensson**: Embodied simulation as off-line representation, 2007.
No 1313 **Zhiyuan He:** System-on-Chip Test Scheduling with Defect-Probability and Temperature Considerations, 2007.
No 1317 **Jonas Elmqvist:** Components, Safety Interfaces and Compositional Analysis, 2007.
No 1320 **Håkan Sundblad:** Question Classification in Question Answering Systems, 2007.
No 1323 **Magnus Lundqvist**: Information Demand and Use: Improving Information Flow within Small-scale Business Contexts, 2007.
No 1329 **Martin Magnusson:** Deductive Planning and Composite Actions in Temporal Action Logic, 2007.
No 1331 **Mikael Asplund:** Restoring Consistency after Network Partitions, 2007.
No 1332 **Martin Fransson:** Towards Individualized Drug Dosage - General Methods and Case Studies, 2007.
No 1333 **Karin Camara:** A Visual Query Language Served by a Multi-sensor Environment, 2007.
No 1337 **David Broman:** Safety, Security, and Semantic Aspects of Equation-Based Object-Oriented Languages and Environments, 2007.
No 1339 **Mikhail Chalabine:** Invasive Interactive Parallelization, 2007.
No 1351 **Susanna Nilsson:** A Holistic Approach to Usability Evaluations of Mixed Reality Systems, 2008.
No 1353 **Shanai Ardi:** A Model and Implementation of a Security Plug-in for the Software Life Cycle, 2008.
No 1356 **Erik Kuiper:** Mobility and Routing in a Delay-tolerant Network of Unmanned Aerial Vehicles, 2008.
No 1359 **Jana Rambusch**: Situated Play, 2008.
No 1361 **Martin Karresand:** Completing the Picture - Fragments and Back Again, 2008.
No 1363 **Per Nyblom:** Dynamic Abstraction for Interleaved Task Planning and Execution, 2008.
No 1371 **Fredrik Lantz:**Terrain Object Recognition and Context Fusion for Decision Support, 2008.
No 1373 **Martin Östlund:** Assistance Plus: 3D-mediated Advice-giving on Pharmaceutical Products, 2008.
No 1381 **Håkan Lundvall:** Automatic Parallelization using Pipelining for Equation-Based Simulation Languages, 2008.
No 1386 **Mirko Thorstensson:** Using Observers for Model Based Data Collection in Distributed Tactical Operations, 2008.
No 1387 **Bahlol Rahimi:** Implementation of Health Information Systems, 2008.
No 1392 **Maria Holmqvist:** Word Alignment by Re-using Parallel Phrases, 2008.
No 1393 **Mattias Eriksson**: Integrated Software Pipelining, 2009.
No 1401 **Annika Öhgren:** Towards an Ontology Development Methodology for Small and Medium-sized Enterprises, 2009.
No 1410 **Rickard Holsmark:** Deadlock Free Routing in Mesh Networks on Chip with Regions, 2009.
No 1421 **Sara Stymne:** Compound Processing for Phrase-Based Statistical Machine Translation, 2009.
No 1427 **Tommy Ellqvist**: Supporting Scientific Collaboration through Workflows and Provenance, 2009.
No 1450 **Fabian Segelström:** Visualisations in Service Design, 2010.
No 1459 **Min Bao**: System Level Techniques for Temperature-Aware Energy Optimization, 2010.
No 1466 **Mohammad Saifullah**: Exploring Biologically Inspired Interactive Networks for Object Recognition, 2011
No 1468 **Qiang Liu**: Dealing with Missing Mappings and Structure in a Network of Ontologies, 2011.

No 1469 **Ruxandra Pop**: Mapping Concurrent Applications to Multiprocessor Systems with Multithreaded Processors and Network on Chip-Based Interconnections, 2011

No 1476 **Per-Magnus Olsson**: Positioning Algorithms for Surveillance Using Unmanned Aerial Vehicles, 2011