CHALMERS
UNIVERSITY OF TECHNOLOGY

Deployment of air defense

An optimization problem using genetic algorithms and high-resolution geospatial data

Master's thesis in Engineering Mathematics and Computational Sciences and Complex Adaptive Systems

Erik Johansson
Sofia Karlsson

Department of Space, Earth and Environment
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Deployment of air defense

An optimization problem using genetic algorithms and high-resolution geospatial data

Erik Johansson
Sofia Karlsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Space, Earth and Environment
Physical Resource Theory
Chalmers University of Technology
Gothenburg, Sweden 2019

Deployment of air defense
An optimization problem using genetic algorithms and high-resolution geospatial data
Erik Johansson
Sofia Karlsson

Supervisor: Håkan Warston, Saab Surveillance
Examiner: Claes Andersson, Department of Space, Earth and Environment

Master's Thesis 2019
Department of Space, Earth and Environment
Physical Resource Theory
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Collage of images representing different aspects of air defense, provided by Saab Surveillance. The center image shows a possible deployment as depicted in this thesis.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Deployment of air defense
An optimization problem using genetic algorithms and high-resolution geospatial data

ERIK JOHANSSON
SOFIA KARLSSON
Department of Space, Earth and Environment
Chalmers University of Technology

Abstract

When planning the deployment of an air defense company, up to date map and terrain information of high enough detail is of importance to quickly find suitable areas of deployment. Saab AB has in cooperation with its subsidiary Vricon developed technology for so called Rapid 3D Mapping from satellite and aerial imagery in order to calculate height, and detect forests, terrain, and buildings with high resolution and great accuracy. In this master's thesis, methods and tools to process and exploit this high resolution data will be proposed. The resulting data will be used in order to optimize the deployment of air defense companies using a genetic algorithm.
An air defense company consists of several types of units. Together they make up the company's ability in battle and are expected to cooperate in order to solve a tactical task or mission. The types of units have different restrictions on their deployment position in order to operate at maximum capacity, and finding suitable deployment sites for the units that maximizes individual and total capacity is a complex optimization problem with many potential solutions. The main types of units considered in this thesis are the sensor unit UndE23, and the fire units EldE97 and EldE98, which are utilized by the Swedish Air Defense regiment.
The preprocessing of the data consists of three steps; identifying positions which are of interest to deploy on, sorting which sites are better than others, and calculating measures for the best sites found which will be used later for the the optimization. Computing the visible area from a position using the high resolution data is very time consuming, and therefore a primary evaluation of the potential points was done using visibility index methods. The best candidates were then evaluated using more detailed methods.
A genetic algorithm was used for the optimization, utilizing elitism and varying mutation rate. The corresponding objective function aims to model both the most important tactical aspects of a scenario, and the most important properties of the units to be deployed.
The results indicate that the optimization method presented gives a satisfying solution, even given a very difficult terrain to deploy in. However, in order to more accurately model the units and tactical aspects of a mission, the objective function and parameter choices could be investigated further. In conclusion, one should use the high resolution data only when needed, as too detailed models could have a very negative impact on computation time.

Keywords: military mission planning, air defense, genetic algorithm, line of sight, big data.

Acknowledgements

We would like to thank everyone who has made this thesis possible, and especially the people at Saab Surveillance in Gothenburg, who have been very accomodating. Furhtermore we would like to express our gratitude to Håkan Warston and Kevin Larsson at Saab, who has taken care of us and provided valuable input during the thesis. We would also like to thank Claes Andersson at Chalmers for interesting discussions and guidance.

In addition, we want to thank our family and friends who have both provided a light during dark times, as well as many great and wonderful moments during our time at Chalmers. We also want to express our gratitude for all the coffee vendors at Chalmers, which have been of great importance for our sanity.

Erik Johansson and Sofia Karlsson, Gothenburg, September 2019

Contents

List of Figures

List of Tables

1
Introduction

A military company is usually part of a bigger organizational structure, such as a batallion, and consists of several types of units; typically sensors, fire units and command units. Together they create the company's capability in battle and are expected to cooperate in order to solve a problem or perform a mission. The types of units have different restrictions on their deployment positions which affect their ability to operate at maximum capacity. Finding suitable positions for the units that enables them to operate at maximum capacity, both individually and collectively, is a complex optimization problem with many potential solutions.
Planning of military missions is performed both before and during the mission. Planning during the operational phase needs to be fast as the current situation can change quickly. There is also a need to have alternative deployments during a mission, as a company needs to be mobile and able to regroup. For example, a reason to regroup could be if the risk of being engaged by enemy units is too high. When finding deployment positions one needs to consider both the conditions related to a specific position for a unit, as well as how the company would perform together in the combination of the positions. Some important aspects when evaluating a set of positions are the size of the covered search space, communication coverage, grouping geometry and the accessibility of the areas.
During the planning, both up to date map and terrain information of high enough detail is of importance to quickly find suitable areas of deployment. Saab has in cooperation with Vricon developed technology for so called Rapid 3D Mapping from satellite and aerial photos in order to calculate height, and reliably classify forests, terrain, and buildings with high resolution and great accuracy. In this master's thesis, methods and tools will be investigated in order to optimize the grouping geometry for military companies.
The thesis is written in collaboration with Saab Surveillance, and is the fourth in a series of theses regarding how to utilize high-resolution data from Vricon.

1.1 Aim

The main objective of this project is to determine how high-resolution geospatial data can be used in a mission planning system for the deployment of an air defense company. This is be achieved by creating a proof-of-concept application for the optimization of such deployments.
During the project different methods for finding optimal deployments are studied, as well as different usages of the high-resolution data and the problems associated

with such large amounts of data.

1.2 Limitations

The application has been limited to the deployment of air defense companies, but could be adapted to other situations and deployments. However the units that are modeled are currently restricted to 3D radar units and fire units using either a fire control radar or an IR seeker in Lock-On-Before-Launch mode. In both cases a free line of sight is needed before the weapon is launched.
The project is based on geospatial data provided by Vricon, consisting of digital surface model data and classification of the terrain. The deployments are limited to static deployments, meaning that the units will not move between different deployments, which is typically the case. It also follows that scenarios where actual enemy engagements or combat occurs will not be considered in the optimization. Other aspects affecting deployment such as weather and bearing of the ground are ignored. The motivation behind simplifications regarding units and communication is described in section 6.3.

1.3 Earlier work

This thesis is the last part of a larger project consisting of four theses, all using the high-resolution data from Vricon. However, it is probable that the series will be expanded, as there are still many unexplored possibilities using the data.
The first project took place during the spring semester of 2018 and was written by Fredrik Aas Isaksen och Kim Nilsen Brusevold at Ostfold University College in collaboration with Saab Surveillance in Halden. They examined and analyzed different algorithms for computing line of sight and viewshed for a given point, which is of importance in order to find good deployment sites for both radar and fire units [16].
A second thesis was performed in Linköping by Mats Nilsson at Linköping University in collaboration with Saab Dynamics, with the goal of reconstructing heights of buildings in the Vricon data using Markov Chain Monte Carlo methods. While his project does not directly relate to the mission planning aspect it is still very valuable, as the height data of buildings can affect possible deployment sites, and therefore must be very precise [19]. However, this is considered out of scope for this thesis, and will be left for a future iteration.
The third thesis was completed by Kevin Larsson at Chalmers University of Technology together with Saab Surveillance in Gothenburg, detailing strategies and possible movements for deployment of an ARTHUR weapon locating radar based on the Vricon data [18]. There are some shared aspects between his work and this project, and in combination with the work done by Fredrik and Kim it has served as a starting point for the use of the map data. In addition, parts of his work regarding preprocessing of the data and the use of genetic algorithms during the optimization process have been used.

1.4 Report outline

Some background and theory necessary for understanding the optimization and the implementation of the project is described in Chapter 2. It describes how the geospatial data works and introduces some geographic coordinate systems. The basic military tactics associated with the deployment of air defense companies are introduced as well.
The objective function used by the optimization algorithm is presented in Chapter 3, and describes how deployments are evaluated based on their positions.
In Chapter 4 the modelling of the application is presented, both with descriptions and details about the programming of the application.
The results are presented in Chapter 5, containing both comparisons of execution times and examples of deployments.
The results and the implementation are discussed in Chapter 6, which also contains a section about how these concepts may be developed in the future.
Finally, Chapter 7 wraps up the discussions and draws some general conclusions about the project.

2
Theory

This chapter introduces some theory and notation used during the project. The first parts present the high resolution data from Vricon as well as some previous projects related to it. Then some neccesary information regarding coordinate systems is presented, as well as the concepts of line of sight and viewsheds. Finally the optimization algorithm is introduced.

2.1 Vricon data

Today, applications using elevation data at Saab Surveillance are mainly based on geographical terrain data with an approximate resolution of 90 or 30 meters, commonly known as DTED1 respectively DTED2 [20]. This data forms a digital elevation model, describing the elevation of the ground without taking buildings and vegetation into account. Now, with the event of commercial sattelite imagery more detailed data becomes available. Vricon [26], a subsidiary of Saab, is one of the world leading vendors of such data. The data, which consists of a digital surface model (DSM) and a digital elevation model (DEM) with a resolution close to 0.5 meters paired with a map of the corresponding terrain classifications described in Table 2.1 [27]. In addition to being more precise, the surface model data is also better suited for line of sight calculations as it is based on surface heights, taking into account the actual heights of buildings and vegetation. The detailed height data in combination with terrain classification gives a good prior knowledge of what to expect of a deployment position in field, making it perfect for mission planning. However, the high resolution increases demand on memory efficiency both related to the methods using the data and for data storage, both of which will be explored in this thesis.

2.2 Geographic Coordinate Systems

There are several coordinate systems used to define locations on earth, and one of the most common ways is to define the position using latitude and longitude. The latitude specifies the north-south position, with 0°corresponding to the equator, and ±90°being the north and the south pole respectively. In a similar way the longitude represents the position on a west-east axis, with 0°being the prime meridian that passes through for example Greenwich in England. The longitudes continue around the earth to ±180°on the other side of the world [13]. Both latitudes and longitudes

Table 2.1: Description of the classes used for terrain classification in the data from Vricon.

Label	Description
Uncertain	Did not meet any criteria
Ground	Grass, dirt, etc.
Low vegetation	0.5-2 m
Medium vegetation	2-5 m
High vegetation	5 + m
Buildings / man-made structures	
Water	
Paved surface	Roads, parking lots etc.

can be specified in decimal degrees, DD or in degrees-minutes-seconds, DMS. In this project we will use decimal degrees.
There exists plenty of models and systems to approximate the irregular shape of the earth. One of the most common systems is the World Geodetic System 1984 (WGS 84) which approximates the earth as an ellipsoid, measuring altitude as the height above the ellipsoid [2]. This allows locations on and above the ground to be uniquely defined based on latitude, longitude and altitude. This is used by most geographical data sources, and is also used by Vricon [27].
As the latitude-longitude representation corresponds to angles on an ellipsoid they are not evenly spaced with regards to distance along all parts of the earth. This refers to the fact that the distance between two longitude lines depends on what latitude is being observed. One can therefore not use this system to compare distances in different areas without using a coordinate transformation.
Instead it is common to use an alternative coordinate system with coordinates more similar to Cartesian coordinates. A common system is the Universal Transverse Mercator (UTM) coordinate system, dividing the world into 60 zones based on longitude, in which coordinates are given as eastings and northings, or x and y coordinates. The UTM system is based on the Mercator projection and the WGS 84 ellipsoid mentioned earlier [4]. Due to Mercator projection distorting landmasses when close to the poles it is only defined for latitudes between $-80°$and $84°$, and it is instead common to use the Universal Polar Stereographic (UPS) coordinate system for the polar regions.
The UTM system is also extended into the Military Grid Reference System, (MGRS) [5], by further dividing the world in 20 zones based on latitude, or 1200 zones in total for non polar regions. The zones are enumerated with numbers for the longitudinal division and letters for the latitudinal division, and inside each zone one can use a Cartesian like coordinate system with easting along the x-axis and northing along the y-axis, both represented in meters.

2.2.1 Curvature of the earth and obscured sight

In this project some additional considerations will need to be made originating from the large distances covered by the given geographical data. Assuming that the earth

is a sphere, but noting the existence of other common models such as an ellipsoid or plane, the curvature of the earth will be a contributing factor to how far the units will be to able see geographically. Given a large enough distance between the target point and the observer the horizon will then obscure some of the target's height, which will be called x. The formula for this is derived by basic trigonometry, whose derivation is left as an exercise to the reader. Considering the measurements presented in Figure 2.1, where the distance to the horizon is denoted a, the formula for the target's obscured height is found in (2.2).

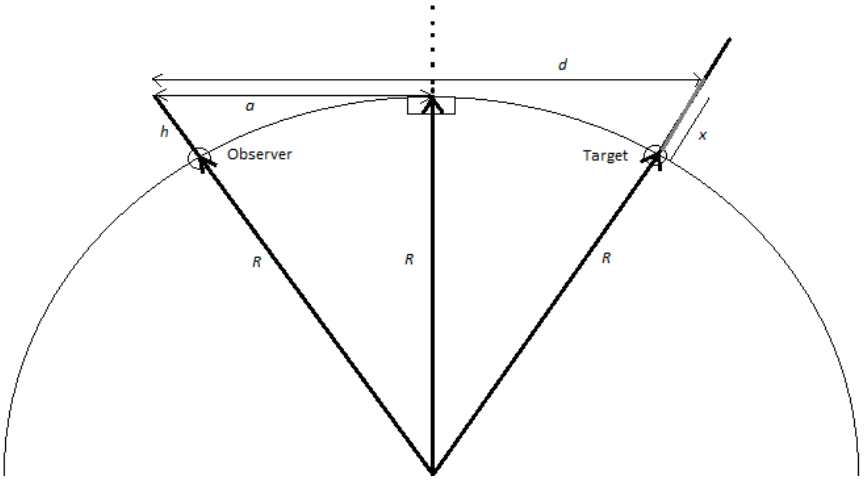


Figure 2.1: Illustration of the calculation of the target's obscured height x. Here, R is the radius of the earth, a is the distance to the horizon for the observer found in (2.2), h is the observer's height above the ground, and d is the total distance between the target and the observer.

$$a = \sqrt{(R+h)^2 - R^2} \tag{2.1}$$

$$x = \sqrt{(a-d)^2 + R^2} - R \tag{2.2}$$

2.3 Introduction to image processing

In order to process the DSM data, either represented as rasters or matrices, some image processing methods will be used. It is common to represent images as matrices where each entry corresponds to the value of a pixel. This value can be in different intervals depending on what the image represents, but in this section the values will be assumed to be either 0 or 1. This is called a binary image, with a pixel taking the value 1 corresponding to a Boolean true value, and a pixel with value 0 corresponding to a Boolean false value. While the image processing tools mentioned below exists for other types of images, they are more intuitive for binary images. A connected component is defined as a group of pixels with the same value that are grouped together, according to some defined neighborhood [11]. There are several definitions of the neighborhood of a pixel, and in this project the eight-neighborhood will be

used. This consist of the four pixels sharing edges with the original one, as well as the four pixels sharing corners with it, creating a square with a sidelength of three pixels, centered at the original pixel. The connected components would therefore be the clusters of pixels that are touching each other with their sides or corners. It is also common to talk about the centroid of a connected component, with which one means the geometric centroid. Connected components are commonly used to remove background noise in images, by identifying small connected components of value 0 and removing them, by replacing their values with 1.
This project will also make use of the morphological operation of dilation, which enlarges the areas containing the value 1 in a binary image. Morphological operations make use of a two dimensional structural element S of some shape, commonly a diamond, a cross or a circle, which is applied to the input area. When performing dilation this element is applied to each pixel of value 1, transforming the surrounding pixels covered by the structural element to the same value [3]. When using the opposite of dilation, which is called erosion, it is instead applied to the pixels of value 0 in the same manner.

2.4 Line of sight, viewshed and visibility

An important aspect of the project will be to determine which deployment sites are better than others for the sensors. The main deciding factor between such sites is how large of an area the sensor can cover. This area is defined by how many points are in line of sight from the sensor. A line of sight is the path between the observer and the point being observed [7]. The target point will generally be visible for the observer if the elevation of the terrain between the two does not go above this line. In order to find the line of sight, the three dimensional positions for the observer and the target point must be known. The most common methods used to find this area will be introduced and discussed in this section.

2.4.1 Viewshed methods

Common methods to find the area a sensor can cover are the so called viewshed methods. A viewshed method calculates the area seen if the observer stands in a point with a specific maximum distance of how far the observer wants to see. The algorithms can be run to evaluate the sight to points either at a specific height above ground, or at a specific height above the sea level. The calculation can be done in different ways, with the most common trade-off being the one between accuracy and speed. The most accurate but also the most time consuming algorithm is called R3. A faster but more approximate version of the R3 algorithm is called R2. Both of these will be presented here.

2.4.1.1 R3

The most simple method to calculate the viewshed is the R3 algorithm. It is non-approximate and calculates the visibility for each point in the grid to every other point within the radius of interest. The algorithm calculates the unique visibility

to the target point by finding if each point in the line between the observer and the target obscures the sight. If a point in this line between them is not crossed exactly, a linear interpolation between the two nearby points will give the height of the crossing, determining the visibility. The area of interest for the observer is partitioned according to Figure 2.2. If the ray of sight crosses a grid line perpendicular to the x-axis the interpolation will be of the two closest points in the x-axis direction and will be called an x-crossing. Similarly, if the ray of sight crosses a grid line perpendicular to the y-axis, the interpolation will use the closest points in the y-axis direction and is called a y-crossing. An illustration of this can be found in Figure 2.3. Note that in octant I, IV, V and VIII more x-crossings will occur, and for the octants II, III, VI and VII there are more y-crossings than x-crossings. This in turn results in that the algorithm will run in approximately $\mathcal{O}(n_R^3)$ time, where n_R denotes the average density of points for the visible radius of the observer [7].

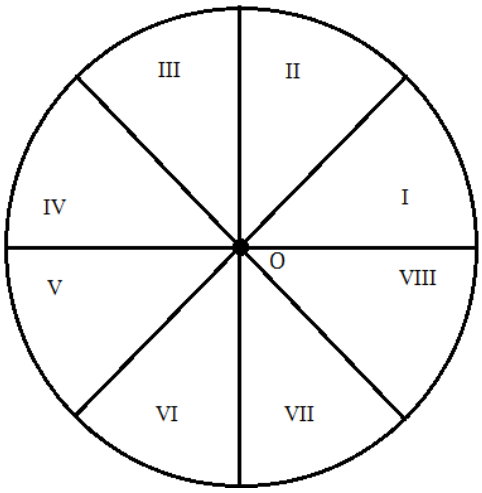


Figure 2.2: Viewshed area of interest partitioned into octants.

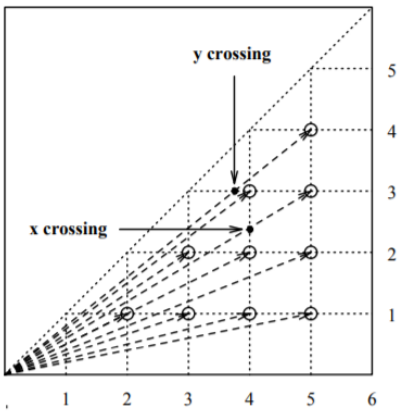


Figure 2.3: Illustration of x and y crossings in the first octant [7].

2.4.1.2 R2

The R2 algorithm was developed to reduce the time complexity of the R3 algorithm with the aim of computing the viewshed in quadratic time complexity $\mathcal{O}(n_R^2)$, where n_R is the point density for the visible radius of the observer. Consider Figure 2.4. Here one assumes that the algorithm has already calculated the viewshed to points a through d, and stored the heights in all crossings to these points. It then approximates the visibility of the points e through g by letting the visibility of these be determined by the nearest stored grid crossing. As the calculations of the visible points move closer to the observer, the crossing points to line of sights for already calculated points further away becomes more closely spaced, thus providing a better and better approximation. Thus, the algorithm calculates a number of line of sights proportional to $2\pi n_R$. The number of locations evaluated along each line of sight with (constant) operations will then be proportional to n_R. Thus the algorithm will have the time complexity of $\mathcal{O}(n_R^2)$ [7].

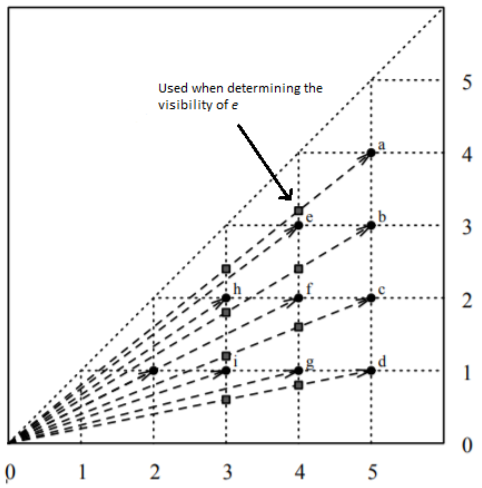


Figure 2.4: R2 approximation in the first octant.

2.4.2 Visibility index methods

Even the more approximate method of computing a viewshed, R2 is quite time-consuming for larger sets of data. Instead one can use an even more approximate method to obtain a preliminary evaluation of the sight at a specific spot, such as a scalar visibility index [12]. A visibility index, or VIX could be computed in a multitude of ways, with the most important thing being that it accurately represents the viewshed at the position, but is much faster to evaluate.

An example of a visibility index could be to sample a large number of points and compute the percentage of them to which there is a free line of sight from the observer.

2.5 Deployment of air defense companies

There are many things to consider when deploying an air defense company, ranging from geographic measures and properties, to pure tactical decisions. In the following section some of the main responsibilities, missions and units related to air defense are described. An air defense company is typically assumed to consist of sensor units, such as a radar, a few fire units and a command central unit (C2). The exact composition may however vary depending on what task is at hand.

2.5.1 Goals and strategies

The targets of an air defense are airborne, and the primary goal is therefore to detect, identify and fight these targets. It should also be a flexible organisation that is able to cover large and diverse areas in several directions at the same time. It is important that the composition of the company is modular, as resources are limited and missions may vary. The company should also be mobile as it needs to change positions regularly.
There are five main instructions that are given to a Swedish air defense company, "Skydda, bekämpa, hindra, stör och luftrumsövervaka", each corresponding to different scenarios. The first action "Skydda" consists of defending a given position or object from airborne targets. With the second instruction, "Bekämpa" , the goal is to limit the activities of the opponent in an area by reducing its resources. The third and fourth objective correspond to preventing and making it harder for the opponent to perform their activities. Finally, the last instruction "Luftrumsövervaka" is to surveil a certain area or airspace. This thesis will focus on the first and last of these instructions, "Skydda" and "Luftrumsövervaka", corresponding to defensive actions.
When protecting an area or an object there are several possible scenarios, and many different tactics to keep in mind. The scenario, and the corresponding suitable actions depend largely on available tactical information about the enemy. Generally there is some information about an enemy available, such as what weapons are used, an approximate position, and what their goal might be. This may be combined with a tactical analysis of the area, containing potential entry ways for airborne targets, important areas to protect and enemy hiding spots.
Based on this information it is common to define a keep out zone (KOZ) around the protected object, in which enemy targets should not be let in. Usually this consist of a radius corresponding to the maximum reach of the weapons the enemy is expected to use, as to not let the protected object be reachable by enemy fire. A consequence of this is that the fire units should be able to fight and destroy enemies before they have a chance to reach the KOZ. In this thesis this zone will be called the engagement zone (EZ). Another common feature is to define a primary target line (PTL) representing a direction where one expects enemies to appear. A PTL is described by a bearing or an interval of bearings with origin at the protected object. A bearing is commonly defined as a clockwise angle measured from the northern angle. Thus, a bearing of 0°points directly north.
The deployment of units is decided depending on the mission, the evaluated enemy

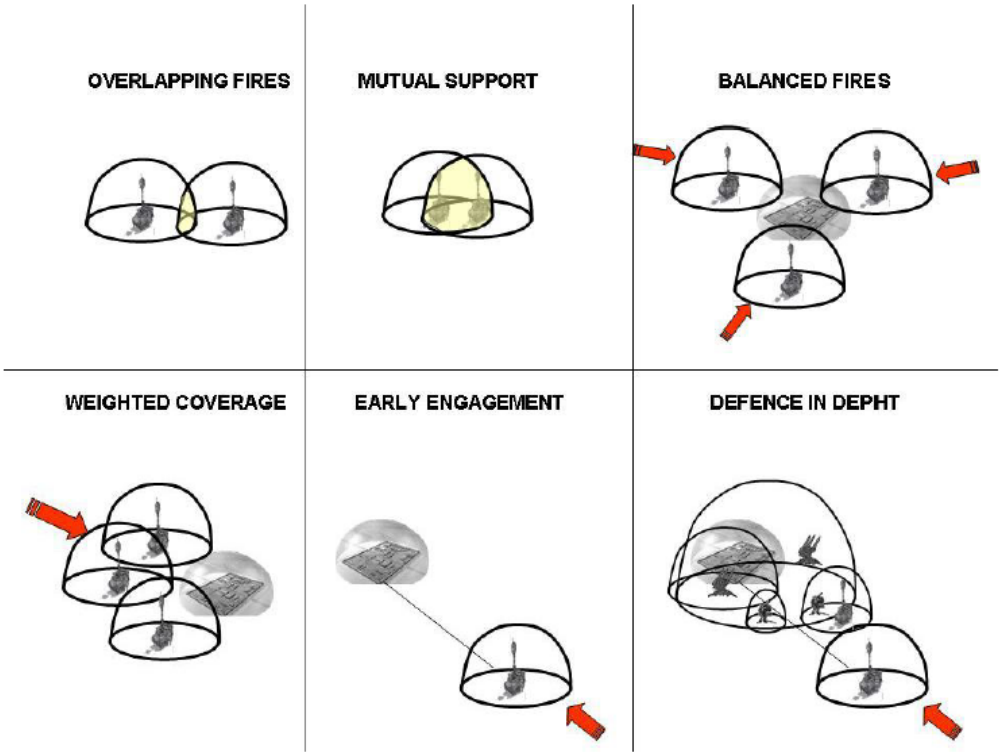


Figure 2.5: The six main cooperation strategies that are used when deploying several similar units used by the Swedish Air Defense regiment [24].

situation and current geographical information, and there are a multitude of possible tactics and deployment principles to choose from. Positions need to be well suited for the unit that is to be deployed there. For example a good position for a radar is one where its search volume is as big as possible with no or few obstructions. However the positions also need to work well together as to allow the entire company to function in unison.
When positioning many units one may decide to favor different cooperation strategies between the units. There are six main cooperation strategies; overlapping fires, mutual support, balanced fires, weighted coverage, early engagement and defense in depth [24]. These images are illustrated in Figure 2.5 The strategies are used and combined depending on the current goal as well as the amount of units that are available.

2.5.2 Units

A company usually consists of a combination of sensors, fire units and a C2 unit. How many units are deployed depends on conditions, availability and the instructions of the mission.

2.5.2.1 Radar units

Air defense companies usually use radar units as the main sensors, which are the main source of awareness about the current situation. There are many models

of radars that can be used with different ranges and strengths, and they are often mounted on trucks or other vehicles for mobility. In the Swedish air defense, Luftvärnet, the main sensor is called UndE23 which is a 3D radar produced by Saab. The UndE23 has a maximum range of approximately 120 km. The antenna of the system can operate either on a height of 8 or 13 meters, depending on surrounding conditions.
Usually the radar units are the first targets during conflicts, as they are vital for the functioning of the defense. Therefore they are commonly considered secondary protection objects with corresponding keep out zones that needs to be protected by the other units in the company.

2.5.2.2 Fire units

There are several kinds of fire units that are used in air defense, with different features, ranges and methods to identify and approach the target. The Swedish Air Defense regiment are currently mainly using EldE97 and will in the future also be equipped with EldE98.
EldE97 is a medium range air defense homing missile system. The missiles have a semi-active homing system, where the target is identified by a sensor unit on the ground, which needs to be able to follow the target during the entire engagement. This method is also called Lock-On-Before-Launch. The missiles are often mounted on a vehicle and can be deployed quickly in many environments.
EldE98 is a new short range missile system introduced to the Swedish Air Defense regiment. It can be used in a similar way as the EldE97, using the homing technique Lock-On-Before-Launch. It can also be fired before identifying the target and then lock on to the target, called Lock-On-After-Launch.

2.5.2.3 C2 unit

The C2 unit is a mobile command central from which all actions of the air defense company normally are directed. The position of the C2 unit is usually chosen somewhat close to the rest of the company as to allow for reliable communication. However this area can be almost anywhere as long as it is reachable and of a certain size.

2.5.3 Communication

In order for an air defense company to be able to accomplish its mission in a coordinated manner it is crucial to create and maintain a common situation awareness among all the units. This consists of the possibility of using both voice communication and sharing data between all units, making sure everyone has access to the same information at the same time.
A prerequisite for the common situation awareness is a communication network of information, to which all units can connect, and in which relevant information can be shared between all network participants. As long as a unit can connect to this network it does not matter if they can communicate directly with the others using another communication method such as radio. If deploying in Sweden it is common

to use Försvarets Telenät, FTN, which is a network that has been created by the Swedish defense and meant to be used for military communication needs. This network covers large parts of Sweden and thus allows for very flexible deployment [10].
Obviously FTN cannot be used if the deployment is outside of Sweden, and in these cases one instead has to rely on creating a network between the units. Then the possibility of communication needs to be taken into account when deploying the company, creating some restrictions. The communication between two units is done through radio using different waveforms and frequencies, and is affected by many surrounding conditions such as humidity, vegetation and specifications of the receiver. The possible range of communication depends on the method that is used, the surrounding conditions as well as diffraction and can be hard to compute in detail for larger areas.

2.6 Basic graph theory

To visualise the communication in a deployment one can imagine the units as nodes in a graph, with the edges representing the communication lines between them. This representation will be used in the optimization algorithm, and therefore some basic graph theory needs to be presented.
Consider a node ν_i belonging to a graph consisting of N nodes. A node is said to be directly connected to another node ν_j if there exists an edge ϵ_{ij} connecting the nodes. Thus a graph consists of several such nodes and the edges between the nodes. One may model this graph as a matrix of the edges ϵ_{ij}, $i, j \in [1, N]$. The matrix will then describe whether there exists an edge between two nodes or if it does not exist one. By letting the first row of the matrix correspond to node one in the graph, and every column be the edge of the node to the other nodes. Thus, if an edge exists between two nodes, the corresponding value in the matrix will be set to 1, if it does not exist it will be set to 0. Repeating this for every node in the graph yields a matrix, E, of size $N \times N$ which is also known as an adjacency matrix. The general matrix can be found in (2.3).

$$E = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \dots & \epsilon_{1N} \\ \epsilon_{21} & \epsilon_{22} & \dots & \epsilon_{2N} \\ \vdots & & \ddots & \vdots \\ \epsilon_{N1} & \epsilon_{N2} & \dots & \epsilon_{NN} \end{bmatrix} \tag{2.3}$$

In order to see if it is possible to traverse from node ν_i to node ν_j in a maximum of one step, one simply looks at the corresponding matrix value ϵ_{ij}. If this value is larger than zero ν_j is directly reachable from node ν_i. It follows that traversing the graph from node ν_i to node ν_j in a maximum number of k steps, one adds a 1 on the diagonal, and then multiplies the adjacency matrix with itself k times obtaining a matrix T which is described in (2.4). Thus, if the target node is reachable in k steps or less, the corresponding matrix element is larger than zero [1].

$$T = E^k \tag{2.4}$$

2.7 Stochastic optimization

As it can be difficult to evaluate and compare different types of deployments, some optimization methods are more suited for such problems than others. Stochastic optimization methods are typically more flexible and dynamic, thus they are a good fit for this particular problem. There are a large variety of methods that fall under the category of stochastic optimization, but in very broad terms one can refer to them as methods for minimizing or maximizing an objective function when randomness is present. The randomness can either come from the problem itself, being present in the objective function or the boundaries, or from the method, usually through random iterations [15]. In contrast to classical methods which mainly are suitable for convex problems or problems with few variables, stochastic methods are able to find solutions to a large variety of optimization problems. Examples of these ranges from problems with the amount of variables varying during the optimization, to having non-differentiable objective functions or an objective function that cannot explicitly be expressed as a function [14]. Due to the stochastic nature of these methods, they also tend to be good at finding solutions to problems containing many local optima, or where the solutions tend to be Pareto optimal, i.e. no solution is the best in regards to every parameter.

2.7.1 Genetic Algorithms

A subset of stochastic optimization methods are genetic algorithms (GAs), which are insipired by Darwin's theory of evolution and natural selection. While there are many variations on the implementation of GAs, the general idea is to encode the variables of the problem into chromosomes representing a solution. A chromosome is also called an individual and is evaluated by an objective function that assigns a fitness. This fitness should typically be maximized. A set of individuals is called a population, which is modified and combined to create new generations and offspring to the earlier population. Thus, one can see the similarities to the evolutionary process both in the method used and in the naming [6].

Some of the most common genetic operators that are used in these algorithms are crossover, corresponding to mating of two individuals, and mutation. While the initial population is obtained randomly, a new generation is usually created through crossover between two chromosomes from the previous generation. The chromosomes are usually randomly chosen, with a preference for chromosomes with higher fitness. The chosen chromosomes are then combined by swapping parts of their encoded variables, resulting in two new chromosomes. These are then altered through mutation and other possible evolutionary operators. This process is repeated until a new generation of the same size as the older one has been created. In order to ensure that the best fitness among the population is monotonously increasing it is also common to implement some element of elitism, saving the current best individual between the generations [8]. The steps of this simple genetic algorithm can be seen in Algorithm 1, and can be altered in many ways as to alternate its behaviour.

Algorithm 1: A simple genetic algorithm

```
begin
    Initialize population randomly
    for iGeneration < total number of generations do
        for all N individuals do
            Decode chromosomes
            Evaluate fitness
        end
        for i < N/2 do
            Choose two individuals
            if Crossover is to be performed then
                Generate two new chromosomes using crossover
                Save chromosomes to new population
            else
                Copy chromosomes of the individuals to new population
            end
            Mutate chromosomes
        end
    end
end
```

2.7.2 Other considered methods

Some other methods considered for the optimization are presented in this section. While they were not chosen in this project they are still useful for other problems. For discussion and justification of the choice of the genetic algorithm they are presented to give some behind the reasoning for deciding on the genetic algorithm.

2.7.2.1 Stochastic gradient descent

Stochastic gradient descent methods are based on the classical method of gradient descent. The methods aim to optimize a differentiable or smooth objective function by iterations influenced by stochastic elements. These stochastic elements could be to calculate the current step length of the iteration on a randomly sampled batch of data. It could also be to let the step length be chosen randomly, usually called gradient descent with momentum. The idea behind the use of stochastic elements is to avoid having the method get stuck in local optima, and it can be shown that they almost surely converge to a global optima if the objective function is convex or pseudoconvex. However, the methods still require the objective function to be differentiable, which many problems cannot fulfill [17].

2.7.2.2 Particle swarm optimization

The particle swarm optimization algorithm considers a set, or swarm, of particles moving around the search space of the problem, where each particle represents a candidate solution. The movement of each particle depends on its current position

and velocity, but also on the particle's best visited position and the swarm's best visited position. Thus, the swarm will behave similarly to a bird flock or school of fish as it searches for a solution to the optimization problem. The algorithm begins with randomizing the starting point and velocity for each particle, then letting the swarm of particles explore the search space as they see fit. After some iterations, the speed of the particles will get lower and lower and the particles will start exploiting the best found positions, and find a candidate solution to the optimization problem. Particle swarm methods are suitable for problems with continuous multidimiensional parameter spaces, such as finding weights for neural networks, signal processing and scheduling [9].

3
Objective function modelling

In this chapter the chosen objective function will be presented and motivated. Details about the implementation are left for Chapter 4. The modelling of the objective function can be split into two parts; evaluating how well the solution solves the problem, and whether the solution has any major flaws and should be considered infeasible.
The importance of these is dependent on some military tactics and restrictions, which in turn depend on the mission, terrain and what type of units are to be deployed.

3.1 Modelling the deployment

A deployment is modeled by a number of units positioned at different points on a map. The position of each unit is evaluated based on sight, using measures such as covered area or visible air space. By evaluating the deployment one obtains a measure of its fitness, which is then used during the optimization.
Both of the fire unit models EldE97 and EldE98 are modeled in a similar fashion, with their range being the only difference. The possibility to use Lock-On-After-Launch for EldE98 is not taken into account in this project.
In order for the air defense company to operate functionally all units have to be able to communicate with each other. This is done through radio communication, which is modeled by a direct line of sight communication between the units. For all units the radio antennas are assumed to have a height of 20 m above the ground. The communication is modeled as a network, which all units needs to be connected to. This network is considered vital to the functioning of the company, and solutions without communication possibility are not considered feasible.

3.2 Choosing an objective function

In a genetic algorithm, each solution is given a fitness where the solution with the highest fitness is deemed the optimal solution to the problem. This fitness can be described in many different ways and differ from problem to problem. Here we will let the fitness for a solution n be called F_n, and describe its general form as in (3.1). The fitness F_n is normalized with the constant C as to be easier to comprehend by the user. We let P_n be a penalty factor for the solution n, making the fitness smaller if the solution does not fulfill some of the required restrictions and limitations. The $F_{n,i}$ are partial fitnesses in the total fitness and describes different parts of the model

to optimize. This could be how well the radar units cover the air volume, or how well the fire units are able to cover the primary target line. Finally, the constants $\alpha_i \geq 0$ describes how much weight is to be given each partial fitness $F_{n,i}$. Thus it is possible to influence the outcome of the optimization by changing the weights and emphasising different partial fitnesses.

$$F_n = CP_n \sum_{i=1}^{I} \alpha_i F_{n,i} \tag{3.1}$$

3.2.1 Partial fitnesses

Every $F_{n,i}$ lives on the interval $[0, 1]$, letting their significance be described by the α_i's. There are in total eight different fitness terms, all contributing to the total fitness of the solution.
The first partial fitness describes how large of a combined area, $A_{h,radar}$, the radar units can see h m above the ground. This height represents a lower limit of what altitudes are considered interesting, and is chosen based on what weapons enemies are expected to have. Thus it follows that the larger area to cover the better the fitness is. This area is compared to the ideal circular coverage area $A_{radar,ideal}$ of the K radars and the fitness is chosen to be the ratio between these, as described in (3.2).

$$F_1 = \frac{A_{h,radar}}{KA_{radar,ideal}} \tag{3.2}$$

The second term corresponds to the observed air volume for every individual radar unit k. It represents how far a radar can see in a straight line for 360 degrees around its deployment site, and finding how large area it can see for a number of elevation angles. These areas for each elevation are then summed, obtaining a measure similar to a volume V_k. The volume for each radar unit is then added together, and the total sum of volumes are then normalized to live on $[0, 1]$ as described in (3.3) by dividing by the number of radar units, K, and the ideal volume V_{ideal}.

$$F_2 = \sum_{k \geq 1} \frac{V_k}{V_{ideal}K} \tag{3.3}$$

The third fitness term describes how large part of the engagement zone's area A_{EZ} is visible to the radar units. The fitness measure for this is found in (3.4), where $A_{EZ,visible}$ is the total area visible by the radar units.

$$F_3 = \frac{A_{EZ,visible}}{A_{EZ}} \tag{3.4}$$

The fourth fitness consists of the combined area $A_{h,fireunit}$ at a height h meters above the ground visible by the fire units. This is normalized to the interval $[0, 1]$ by the ideal circular coverage area $A_{fireunit,ideal}$ for the L fire units. The term is found in (3.5).

$$F_4 = \frac{A_{h,fireunit}}{LA_{fireunit,ideal}} \tag{3.5}$$

The fifth fitness respresents how large part of the engagement zone's area A_{EZ} is visible to the fire units. It is found in (3.6), where $A_{EZ,visible}$ is the total area visible by the fire units, and is very similar to fitness F_3.

$$F_5 = \frac{A_{EZ,visible}}{A_{EZ}} \tag{3.6}$$

The sixth fitness term describes how well the coverage of the fire units overlap at different heights. It aims to make sure that if an enemy aircraft can be engaged on a certain height h_1 close to the edge between two fire units, it should still be able to be within engagement range at another height h_2 larger than h_1. If the terrain is perfectly flat, the half sphere that the fire unit can engage an enemy within covers a larger area at the lower height h_1 than its respective area at h_2. In a more difficult terrain however, where there are hills and mountains, the scenario could be the reverse as the terrain obscures the sight of the fire unit at the height h_1. Thus, for the fitness term we investigate the best intersection area of the intersections every fire unit has with every other fire unit at these two heights of h_1 and h_2. The largest double intersection for the fire unit at the area at height h_1 with the respective area at height h_2 is used for the fitness. This intersection area is normalized to the interval of $[0, 1]$ according to the largest intersection area between h_1 and h_2 for every fire unit present. Now let k be the current of L fire units, j_* the unit with the largest double intersection area A_i with unit k such that $j_* \in [1, L]\backslash\{k\}$. Let A_{k,h_1} be the intersection area between k and j_* at height h_k and A_{k,h_2} at height h_2, thus $A_k = A_{k,h_1} \cap A_{k,h_2}$. Using these notations the fitness is described in (3.7).

$$F_6 = \frac{1}{L}\sum_{k=1}^{L} \frac{A_k}{max(A_{k,h_1}, A_{k,h_2})} \tag{3.7}$$

In the seventh fitness term, visiblity of fire units in different directions are modelled. It aims to promote the deployment points of the fire units with good visibility in the direction of the primary target line, PTL. The main idea is to calculate how much percentual coverage a unit i has in its circular area of visibility divided into eight buckets $B_{i,j}$, corresponding to octants, where j is the index of the bucket. Then, the three buckets closest to the direction of the PTL are found, and the three buckets for the fire unit are then added together with the corresponding buckets for all other fire units in the deployment. Finally the sum is divided by the ideal percentual coverage, which is equal to 1 per bucket in order to bring the fitness term to the wanted interval. The fitness term is found in (3.8).

$$F_7 = \frac{\sum_{i\geq 1}\sum_{j=1}^{3} B_{i,j}}{\sum_{i\geq 1}\sum_{j=1}^{3} B_{i,j,ideal}} \tag{3.8}$$

For the eighth fitness, the double coverage in a specific area A_D for the fire units is considered. The area to be investigated lies in the PTL direction between the KOZ and the EZ and is deemed especially important for the fire units to cover. The fitness measures the largest area requiring double coverage which two fire units can cover at the same time. The visible area being denoted by A_V, and is then normalized by the entire area requiring double coverage. The fitness is described in (3.9).

$$F_8 = \frac{A_V}{A_D} \tag{3.9}$$

3.2.2 Penalty factors

In the objective function (3.1) a penalty factor P_n is included. This factor aims to let the algorithm know how feasible the solutions are and to help guide it to find feasible solutions. Some of the constraints for this optimization problem are more important, for instance that all units must be connected to the same communication network, thus it is of importance that these are fulfilled. If the current solution fails to fulfill many or all of these restrictions this penalty factor will make the total fitness very small. Conversely, if all restrictions are fulfilled, the penalty factor will be set to 1, thus letting the fitness terms be the sole contributing factor in evaluating the total fitness of the current solution. In order to model P_n for a given solution n, we let it be partitioned into factors P_j as found in (3.10), each describing a different constraint for a total of $M = 9$ factors.

$$\begin{aligned} P_{initial} &= 1 \\ P_n &= P_{initial} \prod_{j=1}^{M} P_j \\ P_j &\in (0, 1] \end{aligned} \tag{3.10}$$

The first and most impactful penalty factor P_1 corresponds to penalizing deployments unable to communicate with every unit in the deployment formation. If the units cannot communicate they will not be able to functionally operate together as they do not have a common situation awareness.
The other penalty factors will be less impactful, describing bad behaviour in the solutions but of less importance. The goal of penalty P_2 is to punish deployments where the radar units have large gaps between them, allowing for enemy aircraft to bypass detection. This is applied when radar units are more than a factor times their combined radii apart. The problem is illustrated by Figure 3.1 and Figure 3.2, where the first shows when there exists a gap between the coverage of the radars, and the second without gap.

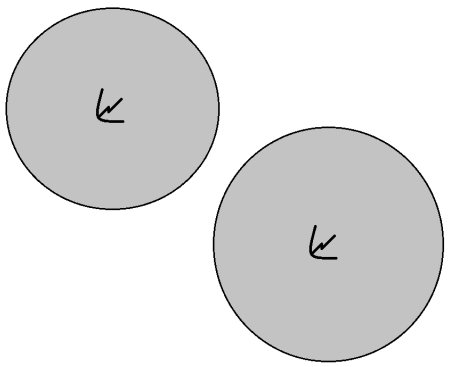


Figure 3.1: Radar units with a gap, allowing an enemy aircraft to bypass detection.

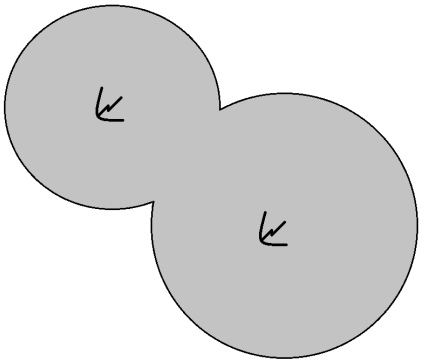


Figure 3.2: Radar units without a gap, makes it harder for an enemy aircraft to avoid detection.

The third penalty P_3 aims to punish the solution if the fire units do not have overlapping coverage with the radar units. The reason for this is that the fire units must see what is visible by radar units in order for the deployment to operate efficiently.
The fourth penalty tries to minimize the spikiness of the radar coverage areas as illustrated in Figure 3.3, as the many spikes allows enemy aircraft to get close to the radar unit without being and possibly take it out. In order to detect these spikes the circumference D_R of the radar area is calculated. If D_R is much larger than that of the ideal coverage, i.e. the circumference of a perfect circle, D_C, a penalty is added. This is because every spike in the edge is contributing to the total circumference. Thus, given a large enough number of spikes, it will differ greatly compared to the ideal version. The penalization is then calculated according to (3.11) and contributes more for a larger difference between the circumferences.

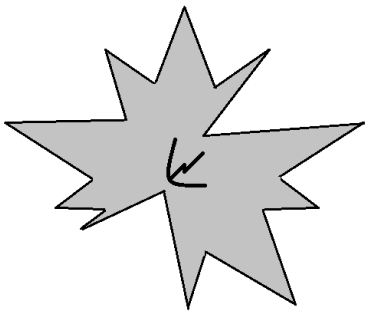


Figure 3.3: Radar unit with many spikes, resulting in poor coverage in some directions.

$$P_4 = \frac{D_C}{D_R} \tag{3.11}$$

The fifth penalty P_5 is applicable when the mission is to protect or guard a certain object or area. It gives the restriction that at least one radar unit and one fire unit must cover the object, otherwise a penalty is applied. This penalty becomes has less impact if at least one radar unit or one fire unit covers the object.
In order for the units to not occupy the same spot, the sixth penalty P_6 penalizes the solution for every unit that is at the same deployment spot as another unit. This penalty is described in (3.12), where N indicates the number of units sharing a spot. Note that the penalty will be equally large for four units sharing the same spot, as for a deployment where two units share one spot and two other units in the same deployment share another spot.

$$P_6 = \frac{1}{N+1} \tag{3.12}$$

The seventh penalty, P_7 describes how good the spot is for communication. It tries to simulate how stable the communication network is if one unit is disabled. Similarly to the first penalty P_1 it investigates if all remaining units can communicate when one unit is completely removed from the network. The penalty gets more impactful depending on how many units are vital for the network to function. This penalty is found in (3.13), where N refers to the number of units that disconnects the remaining network when removed.

$$P_7 = \frac{1}{N+1} \tag{3.13}$$

For the penalty P_8, the aim is to force the C2 unit to stand behind some other units as it is deemed important for it to not be engaged by the enemy. Given a PTL, the penalty is applied if the C2 unit stands closer to the origin of the PTL than all fire units. An example of this is illustrated in Figure 3.4.

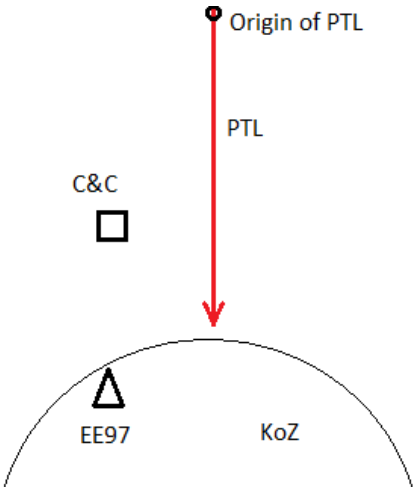


Figure 3.4: The C2 unit is standing closer to the origin of the PTL than the fire unit EE97 does.

The ninth and final penalty factor P_9 works as a complement to P_4 by ensuring that if severe spikiness in the radar coverage area occurs, a solution will still work if the

most vital parts of these spikes are visible by another radar unit. As every radar unit has its own KOZ the penalty applies if no radar in the deployment can see the entire KOZ, i.e. a radar has severe spikes inside its KOZ. In (3.14) this penalty is described, where N refers to the total number of spikes not covered by a radar unit. This is also further illustrated in Figure 3.5.

$$P_9 = \frac{1}{1+N} \tag{3.14}$$

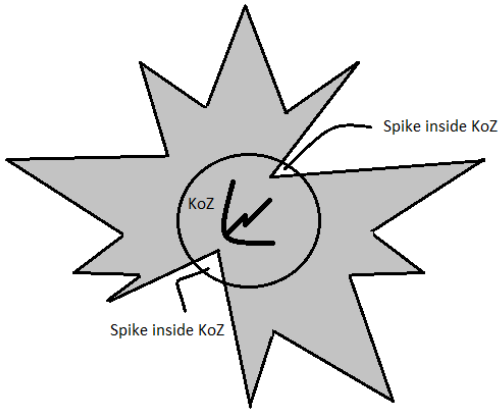


Figure 3.5: A radar unit with two spikes inside its KOZ. If another radar unit cannot see the empty area inside the KOZ, a penalty will be applied.

4
Method

This chapter describes how the mission planning software was implemented and why these decisions were made. All programming was done in Java. Early into the project it became apparent that it could be divided into two parts; finding and evaluating possible sites for deployment of a unit, and optimizing a combination of the deployment sites, evaluating how well the company can be expected to operate at the specified sites. It was decided to create two separate programs for these parts, as one may want to run the optimization algorithm several times using the same set of possible deployment sites.

4.1 Loading and using the map data

One of the goals of the project is to use the data provided by Vricon in an application and to evaluate its usability for mission planning. The data is divided into several files representing smaller rectangles of the area, which will be called maptiles. For each maptile there are several files containing different types of data, such as classification data and surface terrain data. Due to the high resolution of the data, the total size of the files is very large, about 25 GB for a square map of sidelength 50 km. It is therefore impossible to store all the data in the random access memory of most computers. Instead a solution of loading only the data needed at that time is implemented. The chosen solution has the structure of one controller class that keeps track of a large amount of objects representing parts of the map. The class that is used to manage and load maptiles will be called a map controller.

4.1.1 Georeferenced data

The files containing the map data are of format GeoTIFF, which is a TIFF file with corresponding georeferencing information determining its exact spatial references. It includes information such as geodetic system and UTM coordinates of the corners. Information that is of interest are the UTM zone of the map and the coordinates of the corners, both for each maptile and for the combined map. The coordinate of the lower left corner will be the preferred method of identifying maptiles.
The GeoTIFF files are read using `geoTiffReader` from the open source package GeoTools as to obtain the georeferenced data [22]. However, instead of loading the map data to rasters as is done by geoTiffReader, it was chosen to load it into matrices of the type Mat, as used by the computer vision library OpenCV [25]. This allows the use of prewritten methods in OpenCV for computations such as matrix

multiplications and finding connected components, which tend to be quite fast.

The data has a resolution of 0.5 meters, and is divided into maptiles containing 8192 times 8192 pixels, representing a square of sidelength 4.096 km. A file containing DSM data for a maptile has a size around 170 MB, and the size of a classification data file is around 2 MB. While each file is relatively small it quickly adds up for larger datasets. The largest data set used during this project represents a part of Los Angeles and contains 168 maptiles covering an area of around 47 km times 56 km, corresponding to 25 GB of data.

4.1.2 Structure of the map controller

As mentioned, the total size of the Vricon data is too large to be able to load all the data at the same time. Instead the solution is to load the data of the correct maptile only when needed, and to minimize the number of times the data is loaded.

A maptile is represented by an object containing the relevant georeferences as well as paths to the files containing classification and surface data. A maptile is uniquely identified by the UTM coordinate of the lower left corner, which is also used as the origin for an internal coordinate system of the maptile.

The map controller also contains some general geodata for the entire map, as well as a list of the maptiles that the map consists of. It also contains a hashmap that can be used to find the correct maptile to load based on a coordinate. In contrast to a maptile, which uses its own internal coordinate system, the map controller uses a geographic cartesian coordinate system, namely UTM that is presented in Section 2.2. This general coordinate system is what will be used by all other parts of the application, for example as definition of the location of a position.

4.1.3 Downsampling the map

Another approach to manage the large amount of data is to downsample it to obtain a map covering the same area, but with fewer pixels. This is the same as having lower resolution data. For example, by downsampling the Los Angeles dataset to a resolution of 30 m, equal to today's industry standard DTED2, one obtains a GeoTIFF file of size 11 MB. This is much smaller and can be used without memory issues or long loading times.

Since the downsampled data is less exact, it risks missing out on smaller but important features such as single tall trees or buildings. However, it can still be an approximate tool as well as a complement to the high resolution data in order to reduce computation times.

In the applications a very rudimentary approach to downsampling is used. Points in the low resolution data are created at evenly spaced distances based on the high resolution data. The height of the new point is obtained as an average of the eight-neighborhood of the corresponding old point.

4.2 Choice of scenarios

In order to create a functional application, a few of the previously mentioned military scenarios had to be chosen. The choice fell on protecting a certain position, with the possibility of adding a primary target line, a keep out zone and an engagement zone, all of which are available as user inputs. There is also a choice of whether to use double coverage or not, which if enabled adds a fitness corresponding to how well a rectangle is covered by several fire units.

The protection object is defined by a position obtained either through an input of latitude and longitude coordinates, or by a mouse click on a map. The keep out zone and the engagement zone are defined as circles around the protection object, with given radii. The primary target line is defined by the bearing. However, in the application the primary target line will mainly be used as a point on the edge of the map where enemies are expected to come from. The double coverage rectangle is contained within the engagement zone, in the direction of the primary target line. It therefore varies in size depending on the other inputs. An example of a scenario containing all the mentioned features is shown in Figure 4.1.

The geometric shapes are modeled as objects of the class `Geometry` from the package Java Topology Suite, allowing for easy area calculations as well as using methods such as unions and intersections with other `Geometry` objects [23]. For speed of calculation the circles are approximated by polygons containing a few hundred vertices.

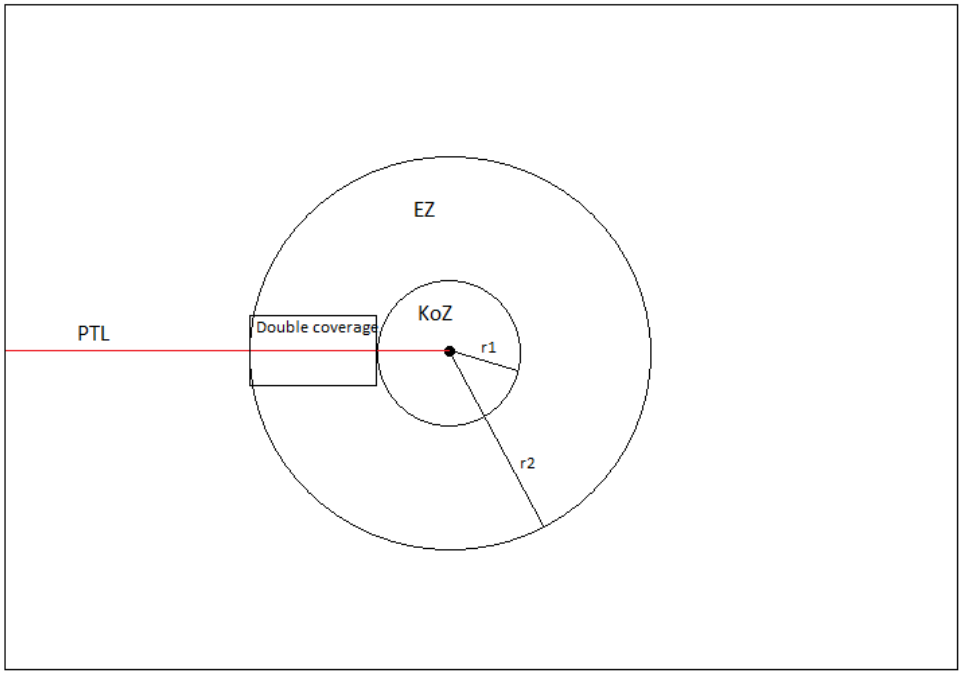


Figure 4.1: Basic figure showing the components of a scenario, with a protection object at given coordinates, a keep-out zone with radius r1, an engagement zone with radius r2 and a primary target line in the bearing 270°. The rectangular double coverage zone is also shown.

4.3 Preprocessing

The goal of the preprocessing software is to find a collection of possible positions for deployment, as well as performing the time consuming computations such as evaluating line of sight and coverage areas of a unit. These positions are to be found using the geographic data from Vricon. In a later version one can also imagine adding other data that might be at hand, such as bearing of the ground, expected weather and layout of the road network.
This application is meant to be used on maps covering a square with sidelength of at least 200 km, as a UndE23 has a maximum range of 120 km. Since the data from Vricon has a resolution of 0.5 meters per pixel there is therefore an enormous amount of points that can be considered for deployment. Thus one of the first tasks of the preprocessing application is to find a limited number of points that are possible to deploy on, and then to further reduce the amount of points until a manageable amount are left. Once that is done, the points can be evaluated for coverage using more exact methods.
The preprocessing is run through a graphical user interface (GUI) where the user can specify what parameters to use and what part of the map to evaluate. Once the application has identified and evaluated a suitable amount of points they are then saved as a text file to be used in the optimization algorithm. The steps of the preprocessing algorithm are presented in Algorithm 2 as well as in the following sections.

Algorithm 2: Preprocessing algorithm

```
begin
    Load map data from GeoTIFF files
    Find possible deployment candidates based on classification and slope
    for each maptile do
        for each smaller square do
            Load suitability data of the square
            Draw Boolean grid on deployment data to divide in smaller parts
            Find centroids of connected components
            for each centroid that is large enough do
                Evaluate close neighborhood around the centroid
                if no very close objects that are taller than the unit then
                    Evaluate using VIX
                end
            end
            Save the centroid with the best VIX value to a list
        end
    end
    for each point with high enough VIX do
        Evaluate the measures
    end
end
```

4.3.1 Finding possible deployment points

The first part of the preprocessing algorithm focuses on finding the positions where it is actually possible to deploy a unit, no matter how good or bad those positions are. There are many things to take into consideration to find such spots which are mainly related to the equipment at hand. A unit usually consists of a few trucks carrying the equipment and personel to the chosen position, which are parked in close proximity to each other. A deployment positon therefore needs to be big enough to accomodate all the equipment as well as be of suitable ground type, both in regards to slope of the ground, bearing and type of terrain. In addition it also needs to be accessible from a road network, preferably without driving offroad for too long or needing to cut down trees. With the exception of bearing and detailed ground type, these conditions can be taken into account using the Vricon data and corresponds to the first step of the preprocessing.
This step consists of using the height and classification data from Vricon to obtain a matrix from which one can identify in which positions it is possible to deploy a unit. The matrix is saved as an image containing binary data, representing if the positions are possible deployment sites.
The classification of a position is allowed to be either ground, low vegetation or paved surface, where low vegetation is allowed as it is believed to be easily removed if necessary. The application also calculates the slope of the ground. The slope is compared to a maximum slope that the units can be deployed on.
As a very simple approximation of reachable points, an extra condition of being within a certain distance of a paved surface or a road is added. This distance represents how far the units are willing to travel in non-optimal terrain. In the application this is performed by using a dilation operation with a given radius on all points classified as road, as described in Section 2.3. It is worth noting that important factors affecting the accessibility, such as slope and type of terrain, are not taken into account in this implementation.
Finally, small connected components introduced in Section 2.3 are removed from the matrix if their area is smaller than what is needed for the unit to deploy on.

4.3.2 Choosing good deployment points

Even with a large portion of the points removed due to not being accessible or possible to deploy on there is still an enormous amount of positions left to evaluate. This part of the algorithm aims to find a subset of the possible deployment points in order to pick out the best choices and to do an approximate evaluation of how good they are.
The map is analyzed one small part at a time in order to choose positions, with the goal of finding the best position in the area. Potential points in the area are found by identifying the centers of the connected components in the data obtained from the previous step. To ensure finding several points for each smaller area a grid of binary zeros is overlaid on the data before finding the connected components. The points that are obtained are compared with each other using the approximate visibility measure VIX [12] and the best proposed point is saved for future evaluation. VIX consists of drawing a large number of random points inside the range of the sensor,

and performing a line of sight calculation from the sensor to the point as to determine whether it can be seen. The VIX value is then obtained as the fraction of the samples that can be seen from the sensor. The line of sight calculation is described in more detail in the next section. Additionally, to speed up the computations VIX calculations are performed on a downsampled version of the map as the line of sight computations tend to be timeconsuming.
Since the VIX calculation is approximate, both in being a method based on samples and in using low resolution data, one also needs to determine that no crucial information in the high resolution data is being lost. Examples of this are large trees being present very close to the proposed position. To model this the closest neighborhood of the point is evaluated using high resolution data, by comparing the height of the ground to the height of the unit, requiring the radar antenna to be higher than all points in a very close neighboring area. If the area is deemed to be good enough the VIX calculation is then performed.
In Java each potential point is modeled as an object, generally referred to as a `DeploymentPoint`. If the VIX value is above zero the point is saved as an object of either the type `RadarPoint` or `FireUnitPoint`, depending on what unit one is looking for. Both `RadarPoint` and `FireUnitPoint` are subclasses to the abstract class `DeploymentPoint`, and are initialized with x and y coordinates of the position and the surface height at that position. The VIX value is also saved into the object, and the objects are stored in a list for future usage.

4.3.3 Evaluating the deployment points

After finding deployment points all over the map the best ones need to be identified and evaluated using a couple of measures. By sorting the list containing the deployment points based on their VIX value a suitable amount of the best points can be chosen. However, the user has the possibility to specify an interval in which the amount of points should lie, as it will affect the computation time.
The measures with which the points are evaluated are inspired by line of sight methods presented in Section 2.4, and were designed to be memory efficient both during computations and when storing the results. The methods also use a combination of high and low resolution data.
After evaluation the final list of `DeploymentPoint`-objects, containing information information about the point and what cases have been used is saved to a textfile, using the `Serializable` interface in Java.

4.3.3.1 Calculation of line of sight

A fundamental part of all the measures mentioned below is the calculation of lines of sight, either between two points or from a given point in a certain bearing and elevation. Both consists of determining how far can be seen along a certain line when stepping along a map with a fixed stepsize. The steps are taken in the coordinate system of the map controller, which in turn loads the correct maptile and surface height if using the high resolution data, or just the correct surface height if using lower resolution data. Since loading maptiles is time consuming the high-resolution

version of the method becomes too slow to be used in all the preprocessing applications, and is therefore combined with its corresponding low-resolution version.
In this application, line of sight between two points is used to determine whether a second point at a certain height above ground can be seen from an observer. If the given point is outside of the map or outside of the range of the sensor it is defined as not being seen. Starting at the observation point the method takes small steps towards the goal and in each step evaluates if the surface height obscures the line of sight or not. The height of the surface is corrected for the curvature of the earth, described in Section 2.2.1 and is naively obtained as the closest raster point to the current exact position.
One can also look at how far an observer can see in a specific direction and elevation. This will be referred to as line of sight along a ray and is computed in a similar fashion. Starting at an observer, the method takes steps in a given bearing and elevation angle, at each step comparing whether the surface height obscures the ray or not. When done the distance that can be seen is returned. If the maximum range is reached or the map ends the ray is considered broken.

4.3.3.2 Sight polygons at an elevation angle

The first measure of the sight of a unit consists of polygons representing how far can be seen around the position in specific elevation angles. This measure is obtained by computing the lines of sight along M rays around the position, with N elevations inside a small interval. The distances are stored as a matrix of size $N \times M$, which is saved into the `DeploymentPoint` object. The distances are converted to points representing where the line of sight in each elevation is broken, and then converted to a polygon representing the area that can be seen. The surface areas are recorded as a vector and will be used as the main measure of how good coverage a point has. The distances are computed on a high resolution map consisting of the immediate neighboring tiles to the one containing the evaluated position as well as on a low-resolution map of the entire area. The distances are then combined in such a way that the low resolution distance in a bearing is used only if the corresponding high resolution ray managed to reach the end of its smaller map. If not, the high resolution distance is used, which is expected to be smaller and more correct than the low-resolution distance. This way the high resolution data is used only in the close neighborhood of the position that is evaluated, while lower resolution is used for larger distances.
In the current implementation a neighborhood of 9 maptiles is used, which means that at least 4 km in all directions is analyzed in high resolution before moving on to the lower resolution data. A change of 1 degree in bearing, at a distance of 4 km corresponds to a horizontal movement of about 70 m, which is much larger than the resolution of DTED2. It is therefore not unreasonable to start using lower resolution data at this distance when using changes in bearing of about 1 degree.

4.3.3.3 Sight polygons at a specified height

The second measure consists of evaluating the area at height h above ground that can be seen from an observer. Starting at an observing point a distance that can be

seen continuously at height h is calculated for N. This is done by computing line of sight between the observing point and a point of the specified height above ground in the given bearing and a distance d, where d is growing with a large stepsize. The largest distance for which all shorter distances are seen is recorded for each of the N bearings. An illustration of this calculation for a given bearing is found in Figure 4.2. These distances are converted to points, and saved as polygons represented as objects of the previously mentioned class `Geometry`.

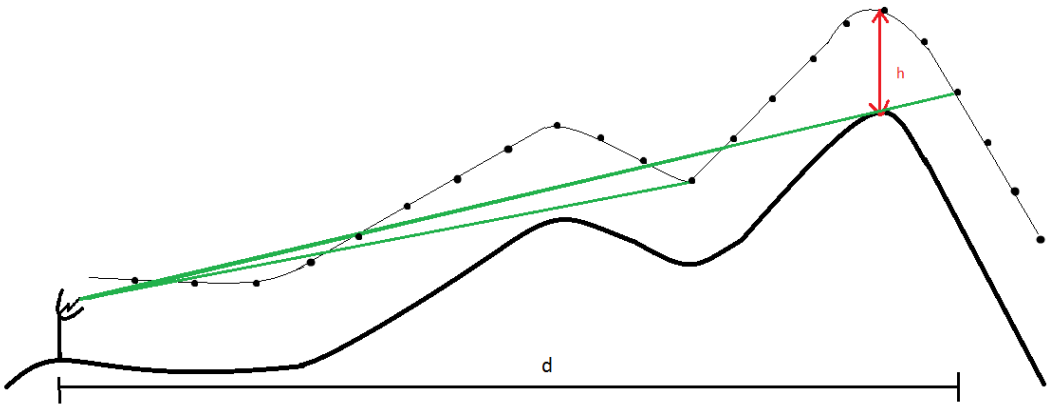


Figure 4.2: Illustration of the calculation of the distance that can be seen continuously at a certain height h in a given direction. The distance d is incremented in steps and the line of sight is calculated to a point at height h above the ground. The value of d, for which the line of sight calculation is obscured, is saved.

In addition, the five worst values of the distance d are saved to a matrix along with their corresponding points. These values are later used by one of the penalties.
This results in a polygon representing a lower approximation of the view, as there may be areas further away that can also be seen. The polygon is a completely connected area, in contrast to a viewshed which does not always consist of a single connected component due to ground being obscured by nearby hills or objects. As the height of interest in military applications is larger than 50 - 200 meters, this approximate representation of the view is likely to be quite similar to the actual view as hills tend to have smooth undulations. This method is only performed on low resolution data.

4.3.3.4 Buckets

The final measurement, which is solely used for fire units, is an approximate value of how good the coverage is in different directions, presented as a vector of numbers between 0 and 1. This is computed by sampling a large number of points in each octant of its range, and computing the fraction of them that can be seen from the observer, in a similar way to how VIX is implemented. These calculations are performed on the low resolution data, and are of importance for the fitness related to the primary target line.

4.4 Optimization

The goal of the optimization is to optimize the deployment of an air defense company. This deployment consists of a predetermined number of units, where their deployment positions are taken from the list of deployment points created during the preprocessing step. The optimization process is based on a genetic algorithm and the deployments are evaluated according to the objective function described in Chapter 3. The optimization is run using a graphic user interface, which also shows the current best results as well as a graph of the convergence of the optimization.

4.4.1 The genetic algorithm

The foundation of the genetic algorithm consists of the classic genetic operators of initialization, decoding, evaluation, selection, crossover and mutation. In order to keep a monotonous increase of the best fitness of the population the elitism operator is added. In this implementation, every individual is created as an object in Java, keeping track of their own chromosome, fitness and relevant operations such as fitness evaluation and mutation. The individual objects were stored as a list of objects in a class containing the main optimization loop.

4.4.1.1 Initialization and choice of encoding

The genetic algorithm maintains a population of individuals, where each individual is encoded with a solution to the optimization problem. An individual is represented as a binary chromosome, where each number is either 0 or 1. These numbers are in a later step decoded and combined in order to obtain the wanted solution. A string of a certain length of binary numbers represents a specific deployment point of either a radar or fire unit. During the preprocessing step, these deployment points were stored as a list. Thus, it is natural that every deployment point encoded in the chromosome represents a specific entry in the list of deployment points. As the list of deployment points is not completely random in the numbering and ordering of the points. This encoding will make it easier for the genetic algorithm to explore potential points which are in close proximity to the current solution. Another way to encode the deployment points is to simply let each gene contain the exact index of a point from the list. This would make the size of each chromosome much smaller. However, the search for a good solution would be much more dependent on good mutations.
At initialization, all individuals are randomly initialized with a sequence of 0's and 1's. Each individual is initialized and inserted into the population one by one until the maximum population size is reached. Because of the randomness in this step, every new genetic algorithm population will have a completely unbiased starting point in order to start exploring the search space.

4.4.1.2 Decoding of the chromosome

The chromosome is binary encoded, hence it is required to decode it before evaluation. We let every deployment point be encoded by n genes. Thus a chromosome

encoding N deployment points contains $n \cdot N = M$ genes. It is then natural to let the first n genes encode the first deployment point, the next n genes encode the second deployment point and so forth. Each deployment point D_i is then calculated by converting the binary numbers describing the specific deployment point in the chromosome to a real integer G_i , as described in (4.1), where the g_j's are the specific binary value of gene j. Then, the integer G_i is converted to the interval of $[0, 1]$, where its value depends on the maximum value the binary encoding can take. It is then finally mapped over to the interval the deployment point indices live on, resulting in the index of the deployment point D_i.

$$G_i = \sum_{j=1}^{n} 2^{-j} g_j \tag{4.1}$$

4.4.1.3 Evaluation

After an individual has been decoded, it is evaluated and given a fitness according to the objective function, see Chapter 3. When all fitness values have been evaluated, they are sorted by size and stored for later use.

4.4.1.4 Selection

After the individuals have been evaluated, selection based on the fitness of the individuals is performed in order to find which genes will spread to the next generation of the population. This selection can be done in many ways, but the method chosen for this problem is the tournament selection.
During tournament selection, K individuals are chosen at random to compete. They are at first ranked according to their fitness values. Then a random number r is drawn and compared to a specific tournament selection parameter p_{tour}, describing the probability of the better individual to win. If the drawn number r is less than p_{tour} the best individual wins, if not, a new number is drawn and then compared to see if the next best individual wins. This process is then repeated until either the better individual wins or only the worst one remains in the competition. The winner is then selected for crossover to spread its genes into the next generation. As it is only the relative fitness values that matter, this method avoids the problem with premature convergence found in other selection methods such as roulette-wheel selection [14].

4.4.1.5 Crossover

In the crossover step the individuals chosen in the selection step will spread their genes into the next generation of the population. This allows genes contained in different individuals to be combined into a new, possibly better deployment solution. The crossover is performed for two chosen individuals with a certain crossover probability p_c. During the first step of the operation, a random number r is drawn. If r is lower than p_c, crossover between the two individuals will be performed. If r exceeds the value of p_c the genes of the individuals will instead be passed down

directly to the new generation. The most basic version of crossover is the one point crossover, illustrated in Figure 4.3, where the crossover point is chosen at random.

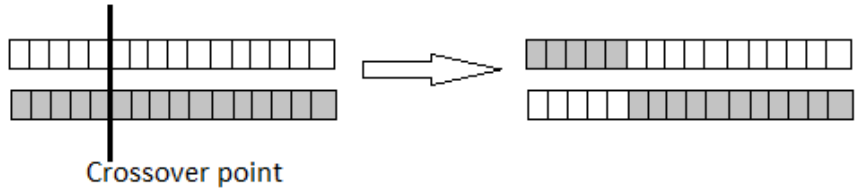


Figure 4.3: A one point crossover for two chromosomes.

In order to better explore the search space and different combinations of deployment points the two point, crossover is used in the implementation. This crossover is illustrated in Figure 4.4, note that the two crossover points are chosen at random.

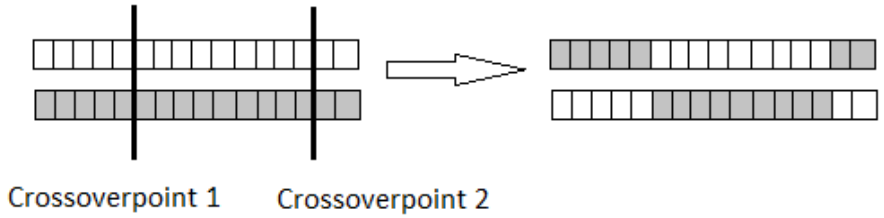


Figure 4.4: A two point crossover for two chromosomes.

After the new individuals have been created through crossover they are inserted into the population of the next generation of solutions.

4.4.1.6 Mutation

An important step in making the individuals explore the search space is to apply the mutation operator on the chromosome. This operation goes through each chromosome and mutates genes with probability p_{mut}, also called mutation rate. As the chromosomes are binary encoded, a gene containing a 0 becomes a 1 and a gene containing a 1 becomes a 0 after mutation. In order to not destroy the genes of the population, the value of p_{mut} is typically set to a low value so that only one gene per chromosome will undergo mutation on average.
As the search may tend to get stuck in a local optimum, a technique called varying mutation rate is used in addition to normal mutation. The essence of this is to increase the rate of mutation if there have not been an improvement to the best solution for a given amount of time. The increase in mutation rate will help the algorithm to explore the search space as the current genes does not seem to contribute in finding a better solution. It is worth noting that the increase in mutation rate only goes up to a certain value, because if it is left unchecked the entire search would converge to a simple random search which is not preferred.

4.4.1.7 Elitism

As selection and crossover is not guaranteed to preserve the best individual in every generation, an elitism operator is commonly used in the genetic algorithm. This operation inserts one or a few copies of the best individuals into the new population without modification. This step also makes sure that the fitness values are monotonically increasing, as the best individual in each generation is preserved into the following one.

4.4.2 Implementing the genetic algorithm optimization in Java

The implementation of the project was done in the Java programming language. In order to make good use of the object-oriented nature of Java, the class `Individual` was created. The class represents each individual in the population considered in the genetic algorithm, and it stores its own chromosome, decoded chromosome, fitness value, and a pointer to the `GAProps` class which will be described in a later section. The `Individual` also contains the functions to compute the fitness value introduced in Chapter 3. The class has two constructors, one for creating a completely new individual which only takes a chromosome size and a `GAProps` reference, and the other requiring a chromosome to be passed down in the form of an integer array. This is to be used if you already have a specific chromosome from an earlier operation in the genetic algorithm, such as crossover, that an individual is supposed to be based upon.

4.4.2.1 The main loop

A class `GeneticAlgorithmOptimizer` was created with the purpose of keeping track of the main loop of the genetic algorithm, as well as finding and storing good solutions to be used by other classes. The class contains the main loop of the algorithm and is initialized by a pointer to the `GAProperties` class. Upon initialization a new population is created consisting of objects of the `Individual` class. It also contains wrapper functions in order to perform the steps of the main loop, such as mutating the population, updating the varying mutation rate and performing crossover and selection. It also keeps track of the best individuals found during the search, both the all time best individuals found, as well as the best individuals in the current generation of the genetic algorithm loop. This information is passed on to the GUI with the help of the `GAProperties` class. An outline for the implemented algorithm can be found in Algorithm 3.

4.4.2.2 Implementation of the fitness measures

As previously described in Chapter 3, the fitness measure for the genetic algorithm consists of partial fitness terms as well as some penalty factors. Several of these terms and factors use some measure of an area which is calculated with the help of the `Geometry` class from the Java Topology Suite package.

Algorithm 3: Genetic algorithm used in the optimization

```
begin
    Initialize population randomly
    for iGeneration < total number of generations do
        for all N individuals do
            Decode chromosomes
            Evaluate fitness
            Store m best individuals in current population for elitism
            Update and store M best individuals of all time
        end
        for i < N/2 do
            Choose two individuals
            if crossover is to be performed then
                Generate two new individuals using crossover
                Save individuals to new population
            else
                Copy individuals of the individuals to new population
            end
            Update mutation rate depending on time since last fitness improvement
            Mutate individuals
        end
        Insert the m currently best individuals directly into the new population
    end
end
```

The fitness value is calculated for each `Individual` in the population. As some of the fitness terms F_i are optional, the fitness terms which have a corresponding α_i equal to zero will be excluded during the evaluation operation of the genetic algorithm.
For the fitness F_1 found in (3.2) it calculates the combined area found at a predetermined height above the ground level which is visible by the radar units. The area polygon for each deployment point is already calculated in the preprocessing step, hence it only remains to combine the polygons for every radar unit present in the wanted deployment. The combined area is then normalized by the ideal area visible on the given map.
The second fitness F_2 found in (3.3) is simply calculated from the area measure for different elevations for a deployment point. These are already precalculated from the preprocessing step and it only remains to add and normalize them.
In order to calculate the third fitness F_3 found in (3.4), the user is required to define the shape of the engagement zone. After finding this area, it makes use of the polygon for a specific height above ground that is visible for the radars, passed down from the preprocessing step. The fitness is then calculated by checking how much the polygon overlaps with the given engagement zone.
The fourth fitness F_4 found in (3.5) is calculated in a similiar fashion as F_1, by finding the polygons for a specific height for the every deployed fire unit. The polygons are

then combined and an area is calculated and normalized.
The fifth fitness F_5 found in (3.6) is calculated just as the third fitness, except using polygons from fire units instead of radars.
The sixth partial fitness term F_6 found in (3.7) describes how well the fire units can cover the air space between two given heights. As it makes use of intersections of the areas, the height polygons obtained from the preprocessing step are used. The largest of the intersections is then used for the fitness. It is normalized by the largest intersected area of the polygons of either the lower or the higher height. This way, the value of the fitness will be lower if the units have a very good overlapping coverage on a lower height above the ground, but barely overlaps on the higher height.
The seventh fitness F_7 found in (3.8) is an optional fitness term. Given that a PTL exists, the fitness will then make use of the sight buckets obtained during the preprocess step. First, it finds what bucket lies closest to the fire unit's angle towards the PTL. It then finds the two neightbouring buckets and adds their value together and normalizes it.
Finally. the optional fitness F_8 found in (3.9) also makes use of the height polygons obtained during preprocessing. It uses them to calculate how good double coverage the fire units will have in the PTL direction. It finds the largest double coverage area between two fire units in the current deployment and normalizes the area by the total area requiring double coverage.

4.4.2.3 Implementing the penalty factors

The penalty factors introduced in Chapter 3 aims to make sure that the genetic algorithm finds and exploits feasible solutions. Some of the penalties are set to a fixed number while others are calculated depending on how bad the solution is.
The first penalty P_1 is generally the most impactful of these. It takes the value 0.001 if the communication network between all units is not fully connected. It is a very simple but effective way of checking if communication is possible between two units. This is investigated by calculating the line of sight between their communication antennas. Thus, if they can see each other they can communicate, otherwise they cannot. The result of this is stored in a so called adjacency matrix. The penalty will then be added if one or more units are not connected to the same network of communication. This is investigated according to Section 2.6, by multiplying the adjacency matrix with itself one time for every unit. If one or more elements of the resulting matrix is zero, the graph is not connected and the solution is not feasible.
The second penalty P_2 penalizes the fitness by a factor of 0.1 if a radar unit is too far away from its closest radar unit neighbor. This distance is computed and then compared to a maximum allowed distance. The maximum distance is set to $1.8(r_i + r_j)$ where the r_i, r_j are the radius of radar unit i and j respectively. The factor 1.8 is chosen in order to have some overlap as it is deemed favourable during operation of the deployment.
For the third penalty P_3, the height polygons of both the radar and fire units are used. It calculates the intersected area visible by both radar and fire units by checking their respective height polygons. Ideally, the radar units can see the entire

coverage area of the fire units. The penalty factor then becomes the fraction between the intersected area and the total coverage area of the fire units.
The penalty P_4 is calculated by investigating the circumference of the radar coverage area. All circumferences larger than the ideal radar circumference is added together. After they have been added, they are divided by the number of radar circumferences added in order to obtain a mean value of the measure. The mean value of this is then compared to the ideal circumference, and if larger, a penalty will be applied. This penalty will be the quota between the ideal circumference and the mean value. Thus, very spiky radar coverage area polygons will make the penalty factor more impactful compared to unspiky ones.
The fifth penalty P_5 simply checks if the object of protection lies inside the area visible by both the radar and fire units. It makes use of the polygons as many other of these factors and results in a penalty if the restriction is not fulfilled. This penalty will be 0.1 if only the radars or the fire units covers the object. If none covers it the penalty will be 0.01, thus more impactful.
For the sixth penalty P_6 the penalty is added if any unit is standing in the same spot as another unit. This is investigated by checking the distances between the units. Every unit sharing a spot with another unit is counted and the penalty is then set according to (3.12).
The seventh penalty factor P_7 is one of the optional penalties. Similarly to the first penalty P_1 it investigates the communication between the units by making use of their adjacency matrix. It investigates the stability of a communication network if one of the nodes and its connections are removed. The process is repeated for every node in the network. Then the communication requirement is calculated as for P_1 according to Section 2.6. After checking the stability for every node in the network a penalty is added according to (3.13), which is more impactful the higher the sensitivity of the network in regards to nodes being removed from it.
The eighth penalty is also one of the optional ones as it requires a PTL. It investigates if a C&C unit is behind at least one fire unit in the PTL direction. It models the PTL as the point of the PTL closest to the edge of the map. Then, the distance to the point is calculated for all units and compared to each other. If a C&C unit is closer to this point than all fire units, the penalty factor of 0.1 will be returned.
The ninth and final penalty models each radar as an object to protect, similar to penalty P_5. It investigates if the radar can see the entire area in its KOZ which is of the same radius as the protected object's KOZ. It first checks if the radar itself can see the KOZ. This calculation uses the five worst points from the height polygon obtained from the preprocessing step, i.e. the points which has the largest spikes. If the distance to one spike is less than the KOZ radius the KOZ is not covered by the radar itself. Then, the point is investigated of whether it is covered by another radar unit. If it still is not covered the penalty will be added according to (3.14).

4.4.3 Running the genetic algorithm

The genetic algorithm is run using a graphic user interface, which also presents results and convergence. Through the GUI the user can specify parameters regarding the mission, such as what units to use and what to protect, choose genetic algorithm

parameters, and choose a predefined military strategy.

Parameters regarding the mission are latitude and longitude of the protected object, radius of KOZ and EZ, bearing of the primary target line, maximum distance of radio communication and amount of radar units, fire units and C&C units that are to be used. Parameters affecting the genetic algorithm that can be changed are number of generations, population size, tournament size, tournament selection probability, crossover probability, mutation probability, elitism size and whether to use varying mutation rate and elitism. All these parameters are set to default values initially, but can be changed if the user wants to.

The user also have some possible choices regarding military tactics, which will determine the weights α_i in the objective function. The primary choice of the user is to select the main strategy for the simulation, which will in turn set the composition of the α_i values. In the current implementation the choice is between 3 modes of optimization. In the mode strong protection the algorithm is focused on creating an impenetrable protection of the engagement zone without holes where an enemy can enter, suitable for a protected object of high importance. The surveillance mode is instead focused on covering larger areas of the map, and might take more risks. The normal mode is a combination of both, which is considered to be similar to a standard deployment. Based on this choice a composition of α_i can be made. The values that are used in the implementation can be found in Table A.3 in Appendix A.

The user can also choose between some optional extra features, namely using double coverage, using a primary target line, and prioritizing stable communications. The double coverage fitness has already been presented as F_8 in (3.9), and is considered optional as it is meant to focus the placement of fire units in the engagement zone, directed towards the primary target line. If the user chooses to not have double coverage, the corresponding α_8 will be set as zero. A requirement for using double coverage is that there is a primary target line defined, which is set through a parameter. The user can however choose to not have a primary target line, for cases when the direction of attack is unknown. The wweight α_7 will be set as zero if the user disables the primary target line, no matter of its value in the currently chosen military tactic. Finally the user can choose to prioritize stable communications, meaning that the communication network of the deployment should be harder to destroy, affecting P_7.

When the user starts the genetic algorithm by pressing run, the parameters in `GAProperties` are updated and in a new thread a new object of the class `GeneticAlgorithmOptimizer` is created. The optimizer then runs in parallel with the graphic user interface, and the best solutions are saved in the `GAProperties` class. The GUI is redrawn when new best solutions are found, showing the deployment and the corresponding area it covers at a height chosen during preprocessing, as well as how much is left before the optimization is finished. The information about the best individual is obtained through `GAProperties`, and the updates about how much is left is done through a property listener attached to `GeneticAlgorithmOptimizer`.

During the optimization the user has the option to cancel the current run, for example if it is deemed to not be good enough. Once the optimization is done

the user can see the best solutions, change scenarios and parameters as well as continue running for more generations on the same population. If choosing to run for more generations on the same population a new `GeneticAlgorithmOptimizer` object is created with the same parameters as before, but with an initial population consisting of the final population from the last run.

4.4.4 The class GAProperties

In order to structure the parameters and constants of the optimization as well as relaying information between the GUI and the optimizer, the `GAProperties` class is created. It contains all the values of the constants and parameters used during optimization, such as the population size of the genetic algorithm, Booleans keeping track of what α_i and a reference to the current `MapController`. When the parameters are set `GAProperties` is responsible for setting the correct values of α_i as well as creating `Geometry` objects representing areas that are used during the optimization, such as the EZ and the double coverage area.

4.4.4.1 Parameters for the optimization

`GAProperties` stores and keeps track of the different parameters and constants used by the optimizer to find an optimal solution. The choice to keep these in a separate class was done due to it being more easily readable and easier to access, as well as saving a lot of space to avoid storing the same information multiple times.
Upon initialization the class gets passed values and pointers to the generic parameters used by the genetic algorithm, the list of different types of deployment points, a pointer to the `MapController`, and parameters to be used when creating a model of the scenario the optimizer will work on. The scenario parameters, for example the coordinates of the protection object and what types of units to deploy, are then combined and a scenario is formed. After a scenario has been created the optimization constants α_i corresponding to the fitness values described in Chapter 3 are set based on what military tactics choices were made.
While the optimization is running, the optimizer continuously updates a list, saved in `GAProperties`, which keeps track of the best individuals of all time found by the genetic algorithm. The best individual in this list is then chosen as the optimal solution for the problem and the entire list is passed on to the GUI to visualize the result of the optimization.

4.4.4.2 Communication between the GUI and the optimizer

When the optimization is running, the main task of the GUI is to update the map visualizer showing the current best solution found. This is done by storing the all time best individuals found during optimization in `GAProperties`, from which the GUI can obtain the solution. In addition to keeping track of the data important to the visualization and optimization, it lets both of these know how the user of the application wants the program to run. For instance if the current genetic algorithm is cancelled or if it should keep running after it is done with its current iterations.

4.4.5 Communication between units

For the implementation of this project, two units are deemed able to communicate with each other if there is a free line of sight between their communication antennas. This is a very simplified assumption as there are many other aspects affecting communication, but it is enough for the scope of this project.
As previously mentioned the calculations for line of sight between two points are quite time consuming when using high resolution data. Instead they are performed on the lower resolution data as to reduce the computation time and speed up the optimization process, as presented in Section 4.3.3.1. This may lead to some faults, mainly in cases with large objects close to either of the points. However, the probability of deploying at a site with large objects that are in the way is low, as this most likely is a bad position to deploy on. Therefore this flaw in the computation of communication is not expected to have any major impact.

5

Results

This chapter begins with a summary of the results regarding the Vricon data. It then continues with results from the preprocessing and images depicting the application. Finally, results from the optimizations are presented, with many images representing potential deployment solutions. All results presented in this chapter were obtained using two computers with identical specifications[1].

5.1 Vricon data

In order to make use of the vast amount of data provided by Vricon downsampling was performed. The largest data set was the Los Angeles data set consisting of 168 maptiles, where the DSM part contains 24.7 GB of information and the classification part contains 349 MB of information. The data sets were downsampled into smaller mapsizes in order for the applications to work, as it is difficult for a normal computer to hold everything in its random access memory. The running times of different scales of downsampling can by found in Table 5.1. It describes the running times to downsample the DSM data to the previous resolution standard of 30 m per pixel to 5 m per pixel, as well as the dimensions of the resulting map. In the same table one can also find execution times for line of sight calculations between randomly sampled pairs of points inside the map area, as to to highlight how much time is saved when stepping between points. In addition, the time to step between the randomly sampled pairs of points can be found in Table 5.2, disregarding if the line of sight is broken or not. These give an upper estimate on the time spent calculating line of sight between points if all points are visible to each other.

[1]Quad-core 2.4 GHz CPU (Intel Core i7-4700MQ), 16 GB RAM, Windows 7 64-bit.

Table 5.1: The execution times, resulting file sizes and time to check line of sight between random points for different sizes of the down-sampled map area, using the DSM data. The number of random point pairs for line of sight was 1000 pairs and a minimum restriction of a distance of 10000 m between them was applied to avoid very short and uninteresting results. The execution time for line of sight is the average time taken of the 1000 point pairs checks.

Resolution per pixel	30 **m**	10 **m**	5 **m**
Time for downsampling	453.831 s	475.036 s	584.203 s
Width in pixels of downsampled map	1575 p	4726 p	9452 p
Height in pixels of downsampled map	1883 p	5650 p	11300 p
Filesize of downsampled map	12 MB	67 MB	382 MB
Line of sight, stepsize of 30 **m**	50 ms	8 ms	6 ms
Line of sight, stepsize equal to resolution	50 ms	16 ms	21 ms

Table 5.2: The resulting execution times for travelling between 1000 randomly sampled pairs of points on a down-sampled map area, using the DSM data. The values corresponds to those found in Table 5.1, but instead of calculating and breaking upon a pair of points not being visible to each other, it kept travelling until the destination was reached.

Resolution per pixel	30 **m**	10 **m**	5 **m**
Stepsize of 30 **m**	0.474 ms	0.460 ms	0.511 ms
Stepsize equal to resolution	474 ms	1318 ms	2781 ms

5.2 Preprocessing

In this section the application for the preprocessing is presented, as well as the results regarding the deployable area and the creation of the points used for the optimization.

5.2.1 The graphic user interface

An image of the GUI for the preprocessing is shown in Figure 5.1. The user can input any parameters they wish to use, but can also use the predetermined parameters. The area that is to be used is marked either by drawing a square on the map with the mouse or by defining the lower left and upper right corners using latitudes and longitudes. Once the user has given their choice of parameters, they press the button "Proceed with preprocessing". Once points have been found they are marked on the map as red crosses, and the user can continue the search with new parameters.

5.2.2 Preprocessing a maptile

There are many steps in the preprocessing, and the following series of images shows the evolution of the data representing possible candidate solutions, also referred to as suitability data of a maptile.

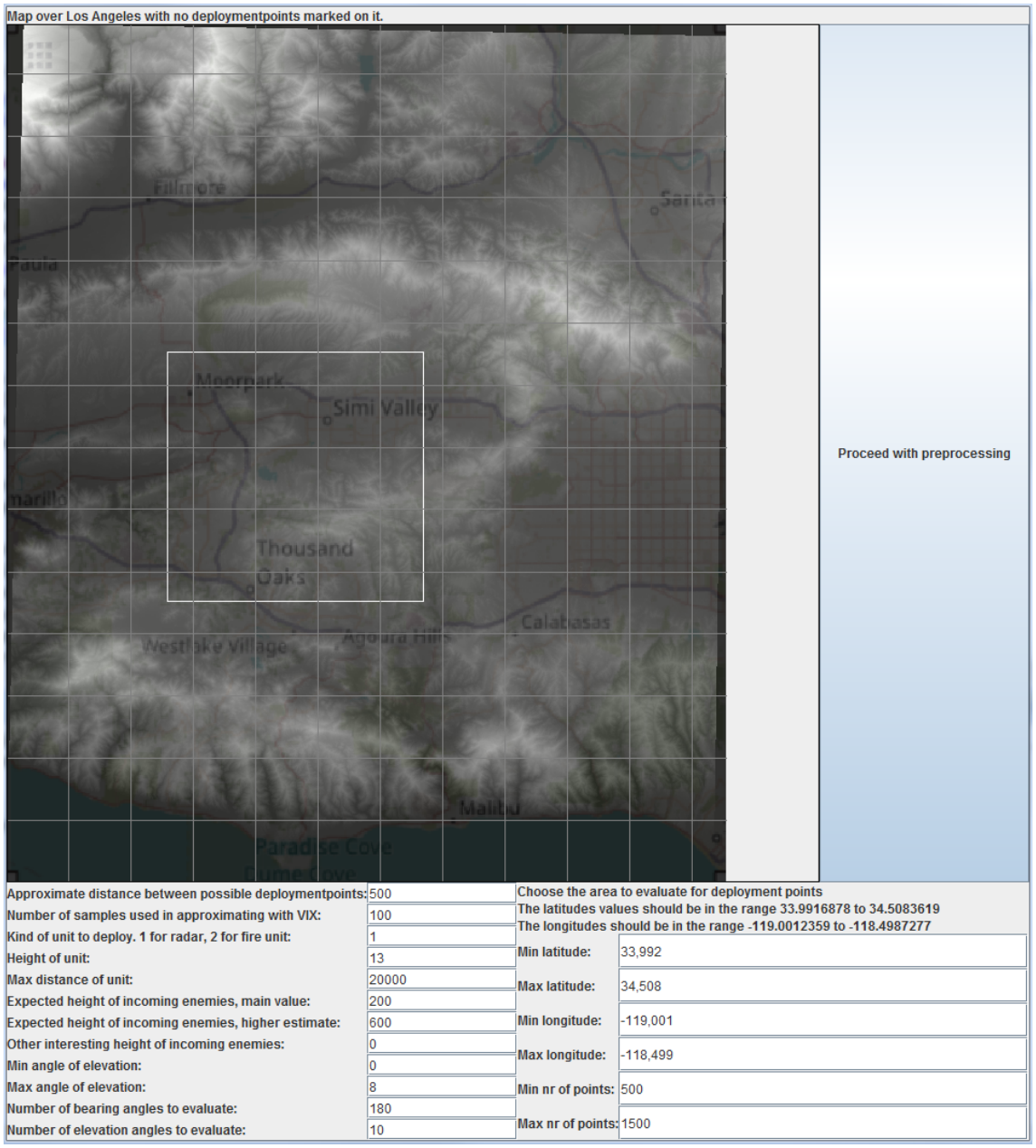
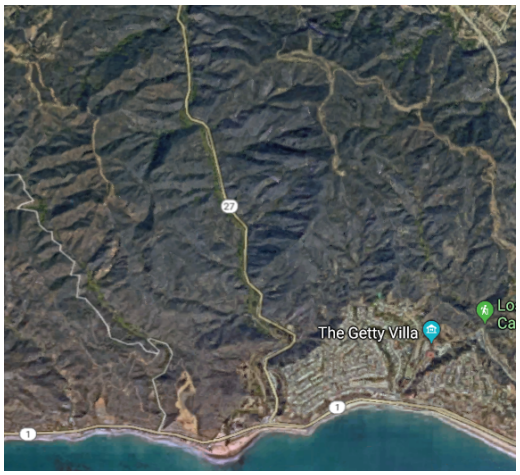


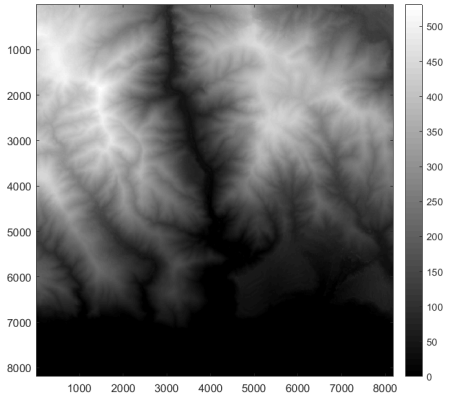
Figure 5.1: Image of the graphic user interface for the preprocessing before running. The marked white square is where the preprocesser will search for points.

In Figure 5.2 the data that will be processed is shown. A satellite image of the area is found to the left and the DSM data is found to the right. One can observe that it is a coastal region, with a small community close to the beach and a mountain area. The maximum height in the area is 531 meters, and the lowest height is -40 meters, with zero being the surface of the ellipsoid used in the WGS84 model.

In Figure 5.3 one can see the three different conditions that are required for a position to be accepted as deployable. The left image shows the areas close enough to the roads as white, set as 300 meters. The center image shows what points have slopes lower than 8°, and the right image depicts all points with suitable classification.



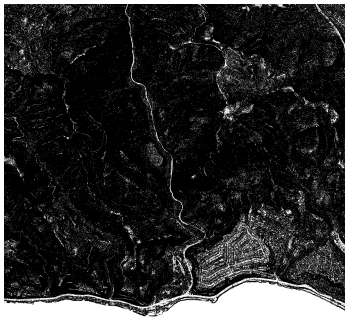
Satellite image of the maptile. Image is taken from Google Earth [21].



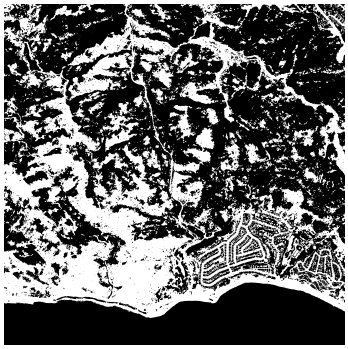
Digital surface map of the maptile.

Figure 5.2: Satellite data and height data of a maptile before preprocessing.

Positions that are close enough to a road.



Positions with slope smaller than 8 °, which is the limit for what is possible to deploy on that is used.



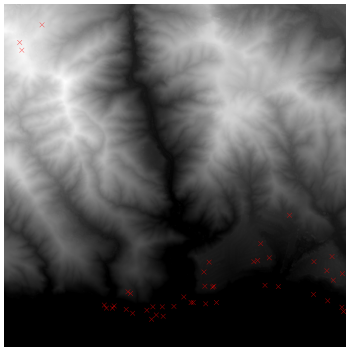
Positions with classification as road, ground or low vegetation.

Figure 5.3: Images showing the three aspects of the analysis of what positions are possible to deploy on. White corresponds to a pixel being identified as deployable based on that condition.

By combining the data in these three images one obtain the final images representing where one can deploy, shown in Figure 5.4. Here the left image shows what is obtained when combining the previous three images. The left image is then processed by removing small connected components that are not big enough to deploy on, resulting in the center image. To the right one can again see the image representing the height data, with the deployment positions that were found marked by red crosses.

Combination of the three measures in 5.3. A point needs to be close enough to a road, with low enough slope and of the correct classification to be possible to deploy on.

Processed version of the combined image, after removing small connected components that are not large enough to deploy on.



Positions marked as deployable, as well the possible deploymentpoints found on the tile.

Figure 5.4: Images showing the final steps of finding the positions that are possible to deploy on, combining the three previous images into one and performing final image processing operations on it.

5.2.3 Deployable area

The data set over Los Angeles contains height and classification data of about 10^{10} points, which is way to much too perform calculations on. The amount of points that are considered for deployment is reduced drastically in the first step of preprocessing. How much the amount of points changes will depend on what map is used, as terrain changes a lot depending on the area. The percentage of points that remain after different steps in the preprocessing for the datasets of Los Angeles and Salisburg can be seen in Table 5.3. With regards to terrain, the two datasets are contraries to each other, with Salisburg being extremely flat and Los Angeles having a lot of mountains. While there is not enough data to draw any conclusions, it may serve as a pointer to what percentages of deployable points one can expect to find.

Table 5.3: Percentages of points that are considered deployable for the Salisburg data and the Los Angeles data. One can see that there is a significant reduction in possible points after only considering a certain distance from a road.

	Salisburg	**Los Angeles**
Approx nr of points	$4.5 \cdot 10^8$	10^{10}
Percent deployable after checking classification and slope	55.17 %	13.98 %
Percent deployable after also considering 300 m from road	36.72 %	10.61 %

5.2.4 Execution times for the preprocessing

Creation of the suitability data for the Los Angeles data set takes 160 minutes, which is slightly less than one minute per maptile. Creating a mapcontroller takes about 17 minutes, and as already mentioned, creating the downsampled map data takes about 7 minutes. However, neither of these calculations needs to be performed each time one wants to run the programs. Instead the mapcontroller and downsampled data are saved to a file from which they can be loaded. All of the execution times mentioned above are expected to grow with the size of a map, mainly due to an increase in maptiles.
The number of deployment points that are obtained from a maptile depend mainly on how large part of the maptile that is possible to deploy on. Therefore the execution time for creating points depends on how large part of the map that is used. Finding potential points using VIX mainly depends on the maximum distance of the unit, the chosen stepsize and on how many samples one wants to make. The points that are used in the results were created in blocks of smaller squares, consisting of four maptiles. For these blocks the VIX part of the computation for one type of unit took a few minutes. The largest part of the preprocessing is however to evaluate the points. This time varies a lot depending on how many points there are, and on what type of unit it is. This is the main reason why a limit on how many points one wants was implemented. The number of points obtained for the blocks mentioned above varied between 200 and 700 points, where the upper limit came in as a user input. For radar units the times to create the points varied between 20 and 80 minutes. For EldE97 these times were 15 and 60 minutes, and for EldE98 they were between 10 and 25 minutes.

5.3 Optimizing

The optimizer is started through the GUI, in which the user can input different parameters for the optimization. An image of the GUI before starting the optimization of a specific mission can be seen in Figure 5.5. In the following sections results regarding the optimization are presented.

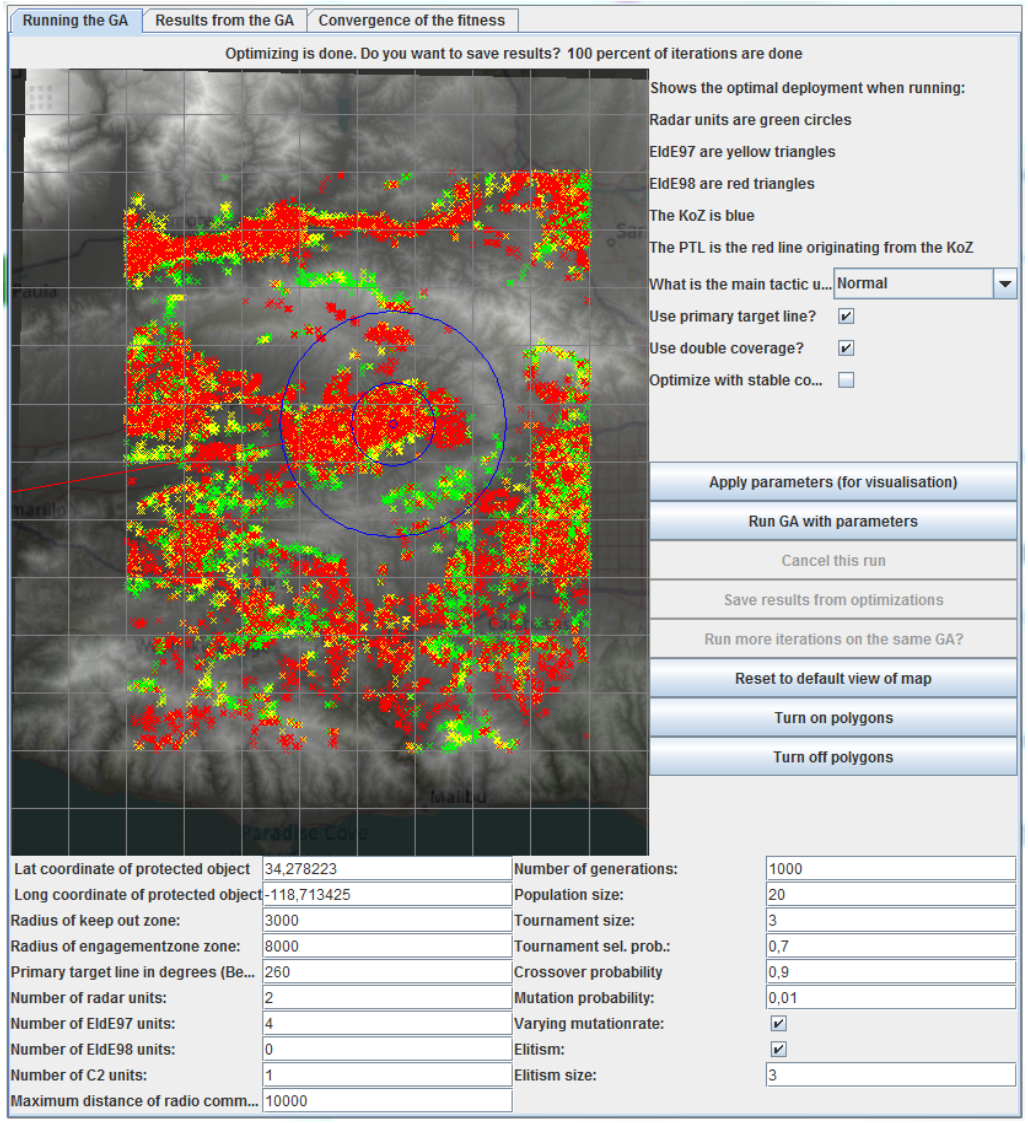


Figure 5.5: The figure shows the graphic user interface before the start of an optimization. The crosses in green, yellow and red represent potential positions for the units UndE23, EldE97 and EldE98 respectively. The KOZ and EZ are shown as blue circles, and the PTL is drawn as a red line. The background is an overlay of the DSM data and a map, on which a grid is drawn. A square in the grid represents a maptile.

As a basis for the results four standard scenarios have been defined, which have been used for the majority of the results. The scenarios consist of a protection object positioned at a given coordinate, with a keep out zone and an engagement zone, as

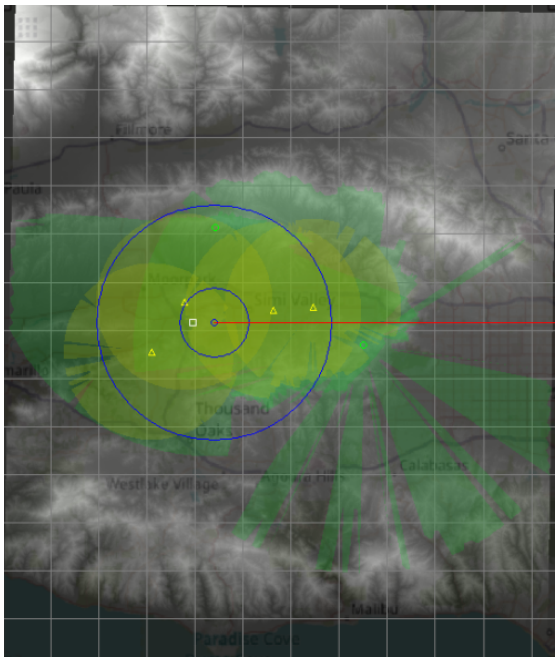
well as a primary target line. The standard configuration of units is to have two UndE23 units and four EldE97 units, however alternative constellations are explored in other sections. The potential deployment points used for the optimization were created using the standard parameters presented in Appendix A, and are located on a rectangle consisting of 80 maptiles in the center of the Los Angeles data set. In some results only a subset of these points are used, which is then specified. In the following sections, the images showing the results of the optimization show the deployments of the UndE23 units as green circles, EldE97 units as yellow triangles, the EldE98 units as red triangles and the C2 unit as a white square. The areas visible for each unit at h meters above the ground is also marked with its corresponding color.

5.3.1 Solutions to the standard scenarios

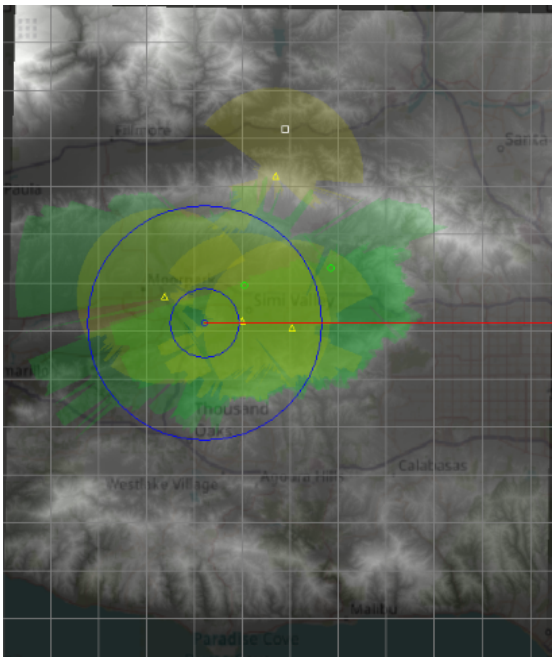
The details regarding the standard scenarios are presented in Table B.1 in Appendix B, and will from now on be referred to as scenario A through D. The results were obtained by running these basic scenarios both with and without the use of double coverage. The results are presented in Table 5.4. Each optimization was run for 1000 generations in the genetic algorithm, and the same scenario was repeated 10 times. Images showing typical solutions are shown in Figures 5.6 through 5.9. These images show the best individual obtained from the optimization and its corresponding deployment. The deployment is shown as a number of polygons, representing the sight at height h meters above the ground, and the positions of the units are marked with green circles for UndE23 and yellow triangles for EldE97.

Table 5.4: Values of best fitness obtained from the four scenarios based on 10 independent runs of 1000 generations. The optimization was made using the normal deployment tactic with a primary targetline, both with and without double coverage. It is worth noting that the fitness can take values from 0 to 10, and that the fitness values cannot be compared when using different optimization parameters.

Scenario	Mean best fitness	Highest best fitness
A	6.6090	6.8141
A with double coverage	4.8831	6.0570
B	3.3839	5.0761
B with double coverage	3.7509	5.1467
C	2.0403	4.4564
C with double coverage	2.2835	3.8743
D	6.1166	7.1989
D with double coverage	5.4131	6.5112

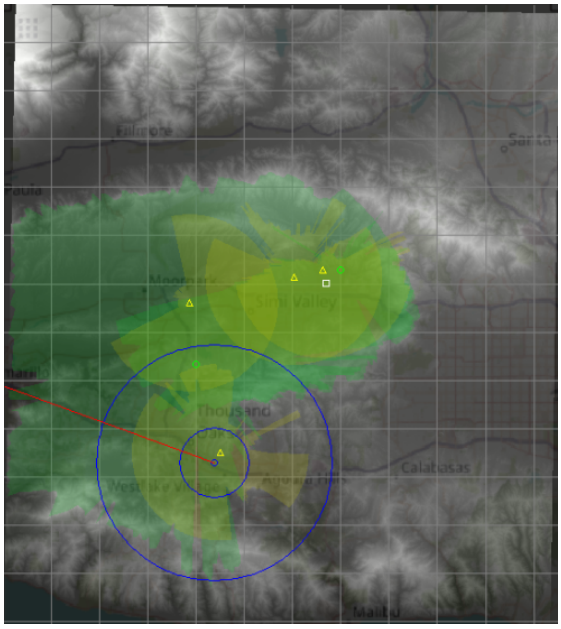


Deployment without double coverage with fitness 6.8141.

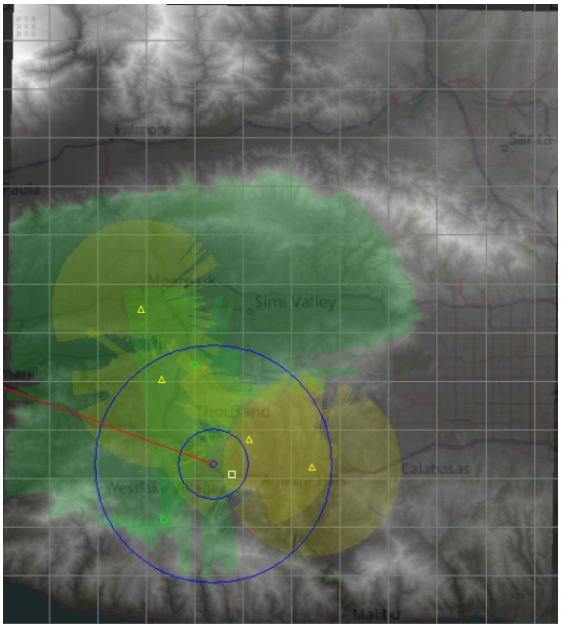


Deployment using double coverage with fitness 6.0570.

Figure 5.6: In these figures the best deployments obtained for scenario A are shown. One can see that both have a decent coverage of the engagement zone for both types of units. However the placements of some units are questionable, and the results could probably have been improved by some more iterations.

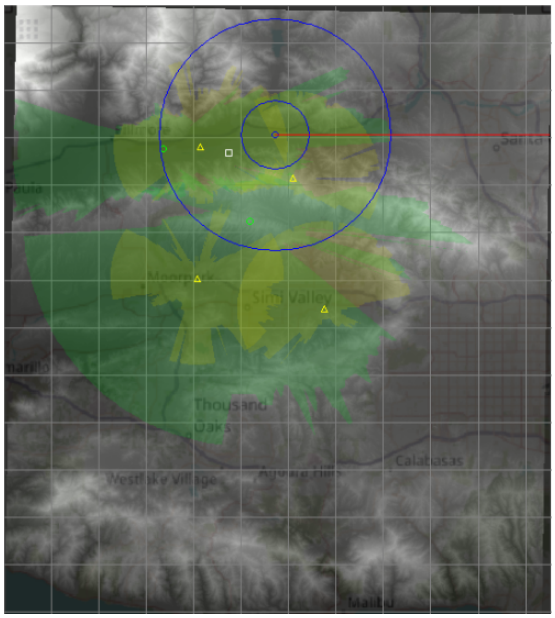


Deployment without double coverage with fitness 5.0761.

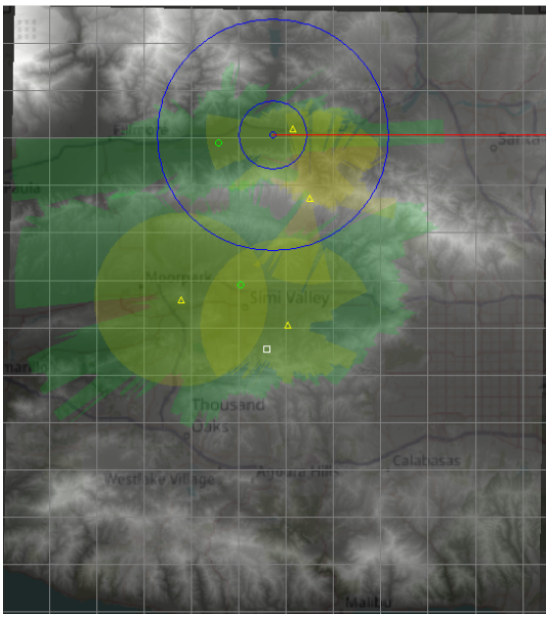


Deployment using double coverage with fitness 5.1467.

Figure 5.7: In these figures the best deployments obtained for scenario B are shown. These deployments have a good radar coverage in the area of interest, however the mountainous landscape and limited amount of points close to the protection object creates problems for the algorithm. Instead it focuses on subparts of the objective function, such as covering large areas even if they are not the most relevant ones



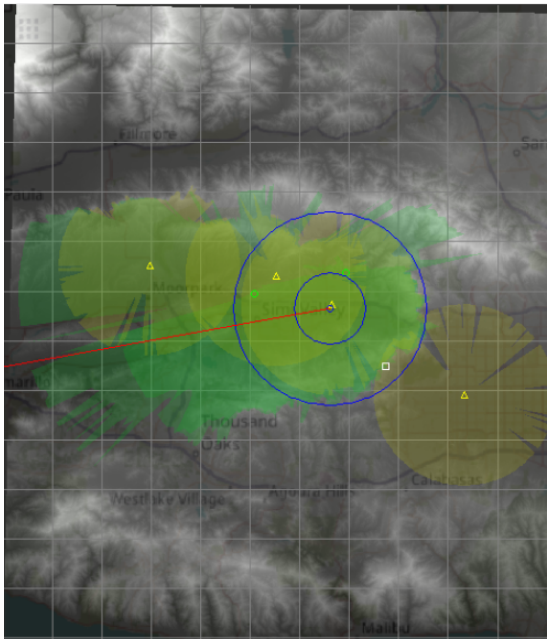
Deployment without double coverage with fitness 4.4564 .



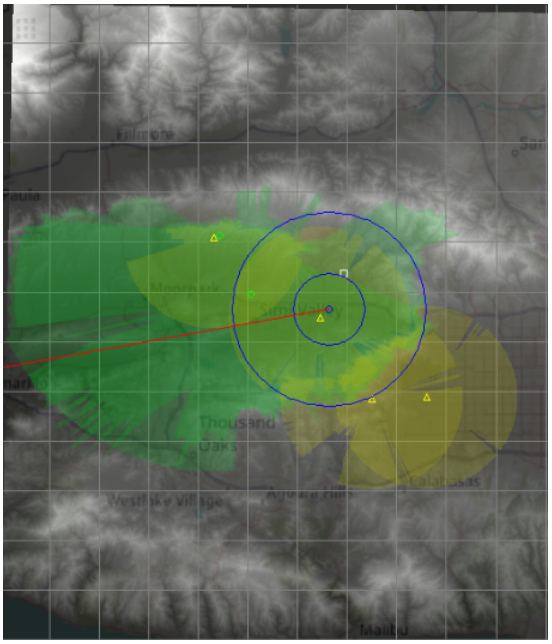
Deployment using double coverage with fitness 3.8743.

Figure 5.8: In these figures the best deployments obtained for scenario C are shown. Here he optimization algorithm has struggled when deploying on a difficult terrain, with limited amount of deployment points. The coverage of the engagement zone is very bad in both cases, as it instead has focused on covering a large area.

Based on the mean best fitness for the scenarios, found in Table 5.4, one can see that the results for scenario B and C are significantly lower than scenario A and D. A reason for this is probably that the protection objects in B and C are located closer to the edge of the map that is considered, causing there to be less points in the immediate neighborhood. In addition, both are located in mountainous areas where there are not as many possible deployment points and the sight is severely limited for the ones that exist. One can see a clear evidence of this in Figure 5.8, where in the left image some units are deployed far below the protection object in an area that is of almost no interest for the scenario.
In several of these runs the optimizer was not even able to find a feasible solution. Because of this only scenario A and D will be used for later results.



Deployment without double coverage with fitness 7.1989.



Deployment using double coverage with fitness 6.5112.

Figure 5.9: In these figures the best deployments obtained for scenario D are shown. Both have good coverage of the engagement zone and the area towards the primary target line. However, one may once again wonder if it would have found a better, more coherent solution with more iterations. It is worth noting that the engagement zone is smaller in this scenario, and can be covered by one fire unit.

5.3.2 Comparison using smaller set of possible deployment points, located around the protection object

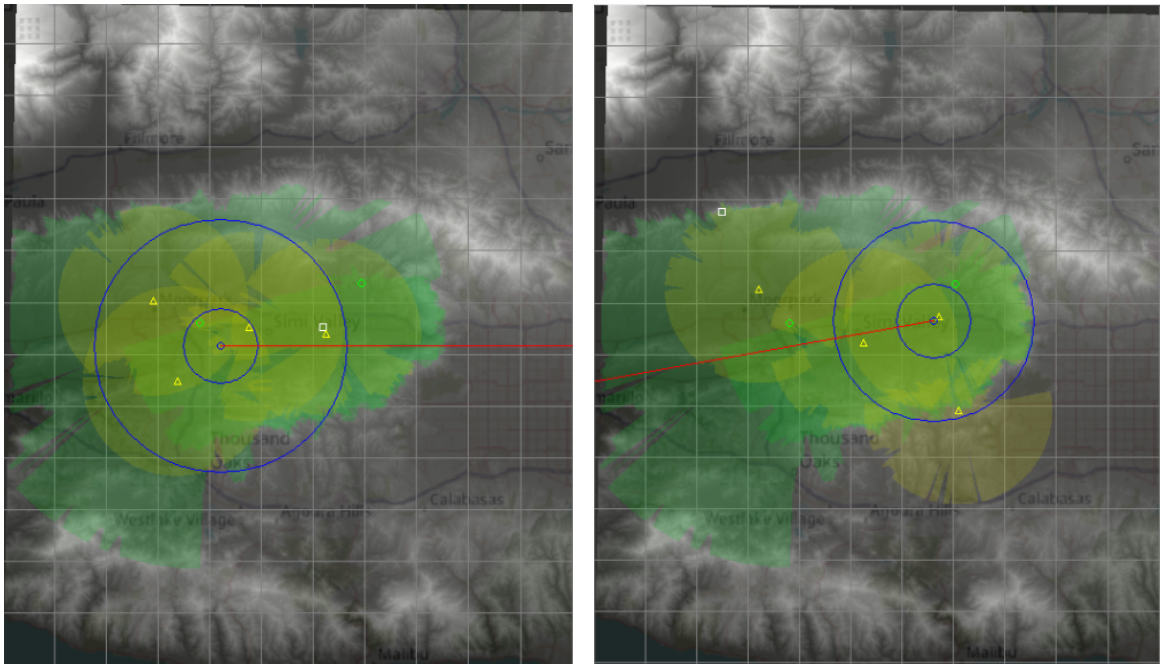
A question one may ask is if the algorithm performs differently when using another amount of possible deployment points spread out on a different size of an area. In Table 5.5 one can see the mean best fitness and the largest best fitness when running scenario A and D on a smaller set of possible deployment points. In this run there are around 4000 points in each category, which are spread out on 36 maptiles close to the protection object.

Table 5.5: Values of best fitness obtained from the first and last scenarios based on 10 independent runs of 1000 generations. The optimization was made using the normal deployment tactic with a primary target line but without using double coverage. These runs were made on a smaller subset of the deployment points, where they are located on 36 maptiles in the proximity of the protection object.

	Mean best fitness	**Highest best fitness**
Scenario A	6.2554	7.3234
Scenario D	5.6262	6.6667

One can see that the mean best fitness values are slightly lower than when using the larger amount of points, however the solutions seem more concise in their de-

ployments. These solutions are often quite similar, with the units being placed in a similar manner and with a more unison formation. Some examples of this can be found in Appendix B.2. Images representing the best deployments obtained during the 10 runs can be seen in Figure 5.10.



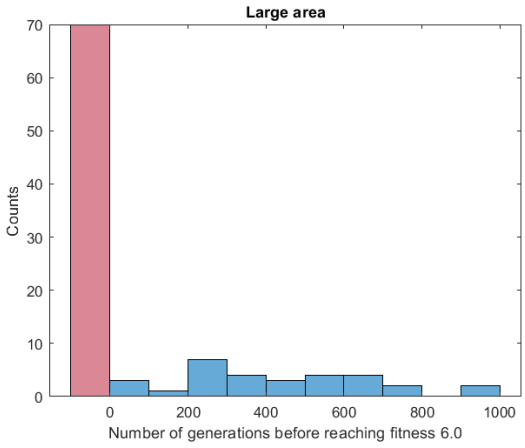
Deployment for scenario A with fitness 7.323.

Deployment for scenario D with fitness 6.667.

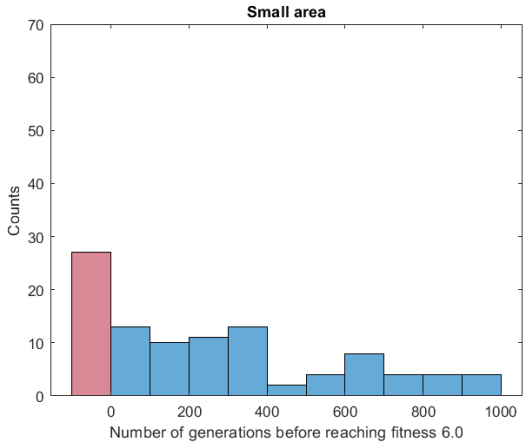
Figure 5.10: Deployments with the highest fitnesss on the small map containing 36 maptiles for the scenarios A and D, obtained from 10 runs with 1000 generations.

A second comparison using the different sets of points was made as to evaluate whether the deployments on the smaller map were more similar to each other than the deployments on the larger map. This was performed on scenario A over 1000 generations. 50 runs were made on each set of points, recording the highest fitness value that was obtained during the optimizations. Using the large set of points the mean value of the highest fitness was 5.5709 with a standard deviation of 0.8483. Using the small set of points the mean value of the highest fitness was 6.2915 with a standard deviation of 0.7473, which is a higher mean and a lower standard deviation than for the large set of points. It therefore seems as if the results using the smaller set of points are better, at least when evaluating over 1000 generations.
In addition to evaluating the similarity of the deployments on the same map a test comparing the number of generations needed to reach a certain fitness value was performed. Just as in the previous test, scenario A was used on a maximum of 1000 generations, and performed 100 times. If the optimization reached a fitness value above 6.0 it was stopped, and the current generation was recorded. Histograms comparing the results are shown in Figure 5.11, where the first bin in red corresponds to the run never reaching the required value. One can see that not only is there a significantly larger part of the runs that never reach this fitness on the large map, but the ones that do also tend to do it at later generations. As the smaller map is

a subset of the larger map this indicates that the large map requires longer time to converge.



Histogram of number of generations needed to reach a fitness of 6.0 for scenario A on the larger set of points. The first bin, in red corresponds to runs that had not reached the fitness in 1000 generations.

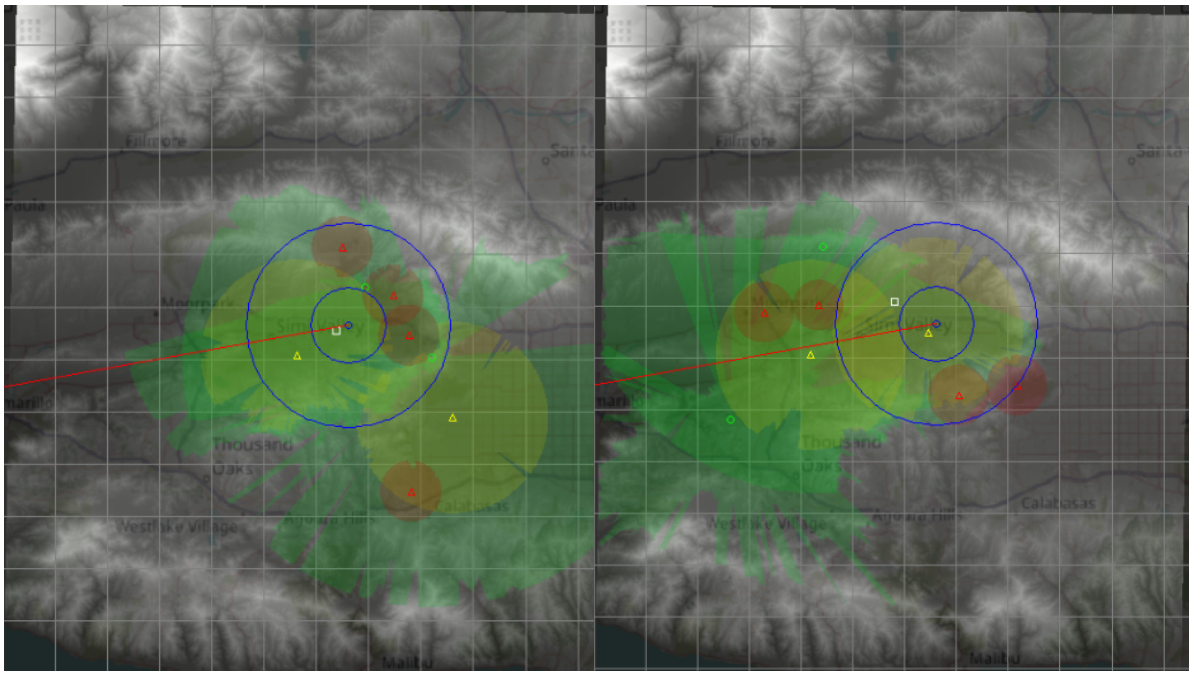


Histogram of number of generations needed to reach a fitness of 6.0 for scenario A on the smaller set of points. The first bin, in red corresponds to runs that had not reached the fitness in 1000 generations.

Figure 5.11: Histograms of number of generations needed to reach a fitness of 6.0, with the first bin corresponding to runs that had not reached the required fitness in 1000 generations. One can see that there is a much larger proportion of runs on the large set that do not reach this value than on the small set.

5.3.3 Comparison of scenario D for a different composition of units

The previous results were obtained using a standard deployment of two UndE23 units and four EldE97 units. However there are other combinations of units that are of interest to deploy. One example is when using two UndE23 units, two EldE97 units and four EldE98 units. These were deployed on scenario D, and the process was repeated 10 times for 1000 generations. The two best deployments are shown in Figure 5.12, and more examples can be found in Appendix B.3. One can see that the resulting deployments vary a lot in composition and sometimes places a unit far away without any apparent reason. Another prominent behaviour is that the deployments tend to use the smaller fire unit EldE98 to fill in holes in the engagement zone, as to better protect the keep out zone. It is worth keeping in mind that these results were obtained using 1000 generations, which might not be enough as there are more units to deploy and optimize for, and that the large lists of points over 80 maptiles was used.



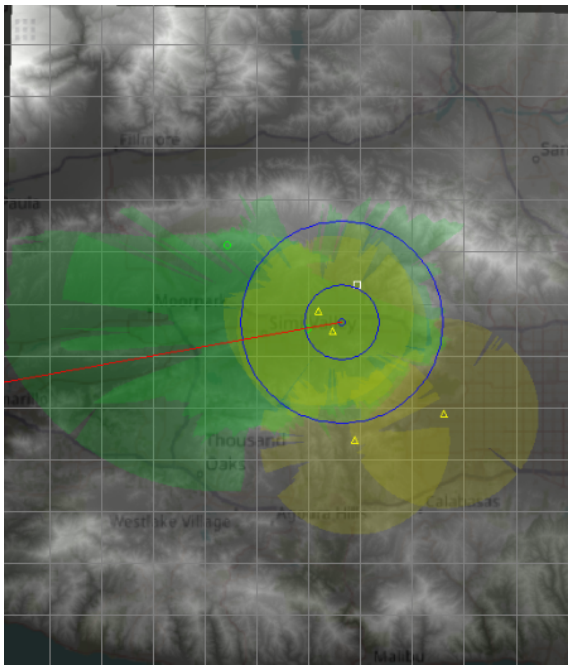
Best deployment using the alternative composition of an air defense company. The fitness of this deployment is 6.9208.

Second best deployment using the alternative composition of an air defense company. The fitness of this deployment is 6.4084.

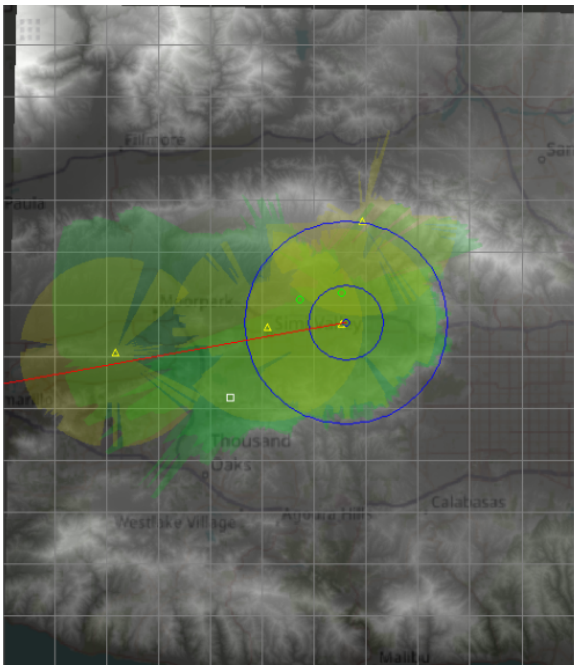
Figure 5.12: Comparison of the two best deployments on scenario D. The deployment consists of two UndE23, two EldE97 and four EldE98.

5.3.4 Comparison of the same scenario using different tactics

Three tactical choices were implemented in the model, consisting of a default mode, a defense mode and a surveillance mode. These modes determine what values the weights α_i are set as for the partial fitness. Two typical deployments using each tactic are shown in Figures 5.13 to 5.15. One can see that the defensive mode has a tendency to place a couple units inside the engagement zone, making sure it is completely covered. This is not as common for scenarios using the surveillance mode, which instead spreads out the deployment, covering as much area as possible. Often this can cause odd-looking deployments which seem randomly placed.

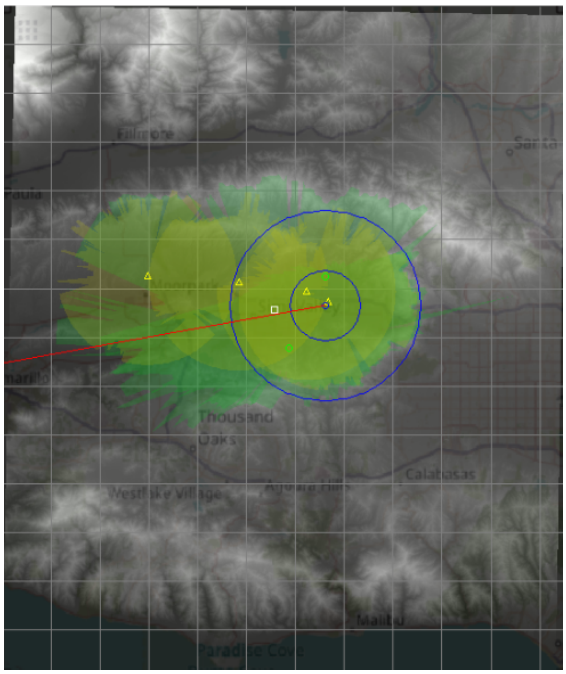


A result of running the default mode on scenario D.

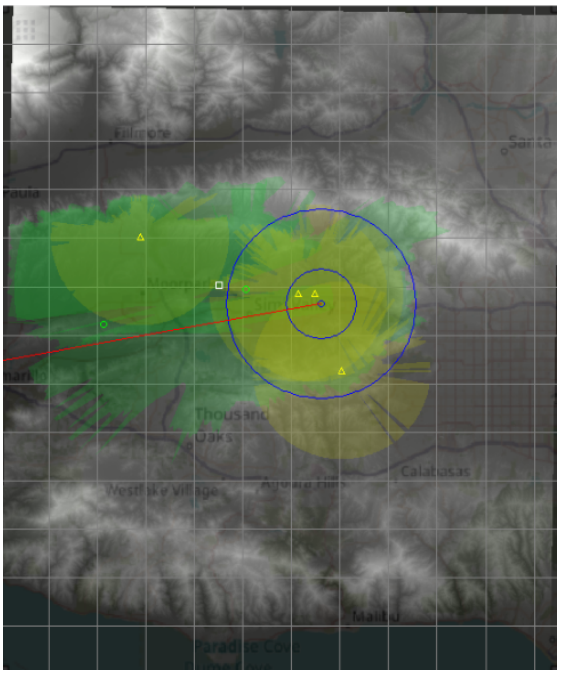


Another result of running the default mode on scenario D.

Figure 5.13: Two typical results of running the default tactical choice on scenario D. The units are slightly spread out, but still have good defensive coverage.

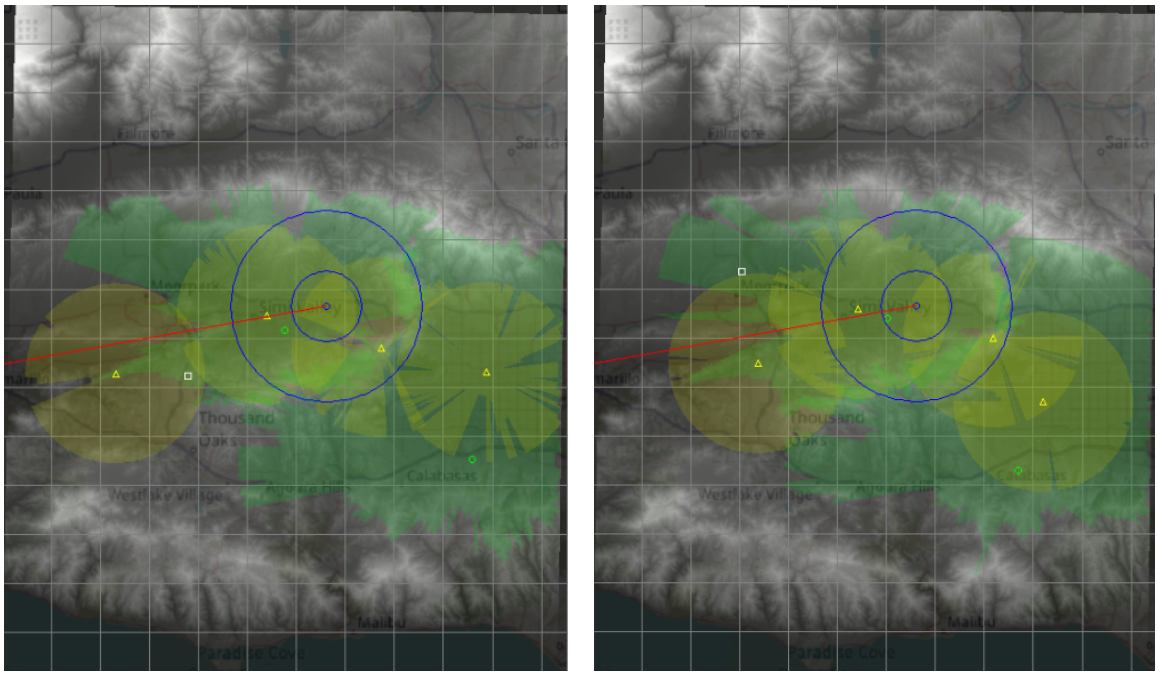


A result of running the defense mode on scenario D.



Another result of running the defense mode on scenario D.

Figure 5.14: Two typical results of running the defense tactical choice on scenario D. Here one can see that at least two fire units are deployed very close to the protection object, giving a very good coverage of the engagement zone.



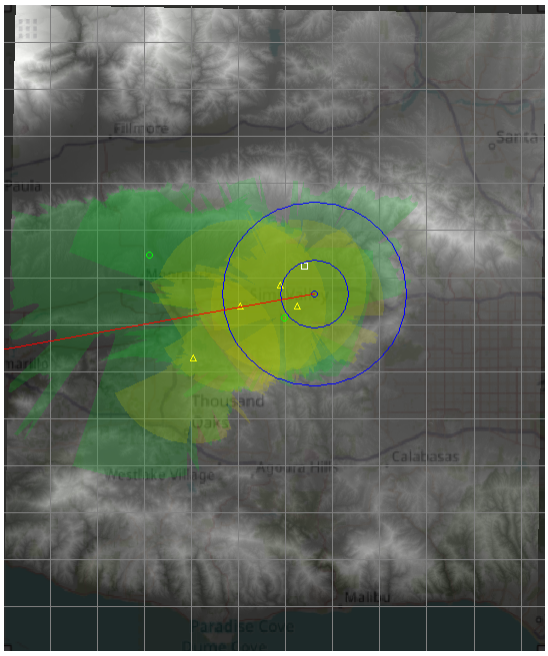
A result of running the surveillance mode on scenario D.

Another result of running the surveillance mode on scenario D.

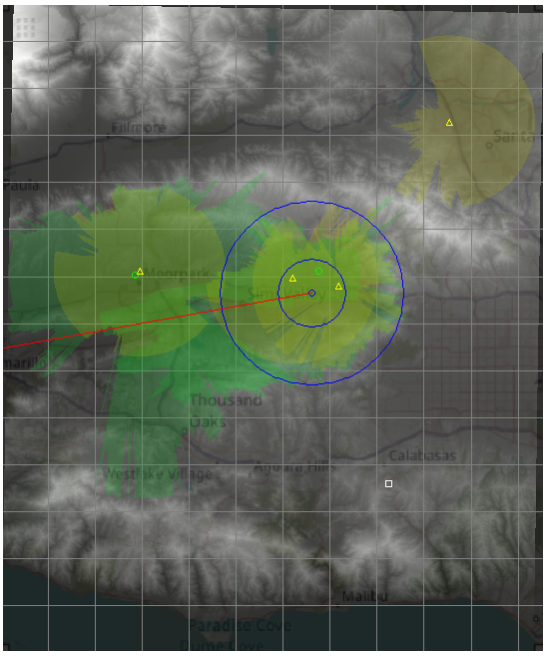
Figure 5.15: Two typical results of running the surveillance tactical choice on scenario D. The units are quite spread out, and cover a large portion of the map.

There was also found to be a difference in how the optimizer searches for solutions between the defense mode and the surveillance mode during the first generations. The initial generation is of course completely random, but it is clear that the defense mode favors solutions covering the engagement zone. This quickly causes most of the units to deploy close to it and the solution to become feasible. During later generations it keeps on refining this solution. The surveillance mode however takes longer time to find a feasible solution due to favoring very spread out deployments, making it harder to fulfill the communication requirement. Examples of typical deployments after 10 generations can be seen in Figure 5.16.

In addition, there seems to be a subtle difference in the deployment of the defense mode depending on the size of the engagement zone, which is considered very important to protect in this mode. In scenario D the engagement zone has the same radius as the EldE97 unit, allowing for all of it to be covered by a single unit. In Figure 5.14 one can see how one or two units are often placed around the protection object, and the other ones further away. However, when using an engagement zone that is bigger than the radius of the fire units it seems to be more common to place three of the fire units in a triangle, surrounding the protection object. An example of this is shown in Figure 5.17.



Deployment using the defense tactic on scenario D after 10 generations, with a fitness of 2.988.



Deployment using the surveillance tactic on scenario D after 10 generations, with a fitness of 0.003.

Figure 5.16: Deployments on scenario D after 10 generations. One can see that the deployment using the surveillance tactics are much more spread out, whereas the defensive tactic has already found a more closely organised formation.

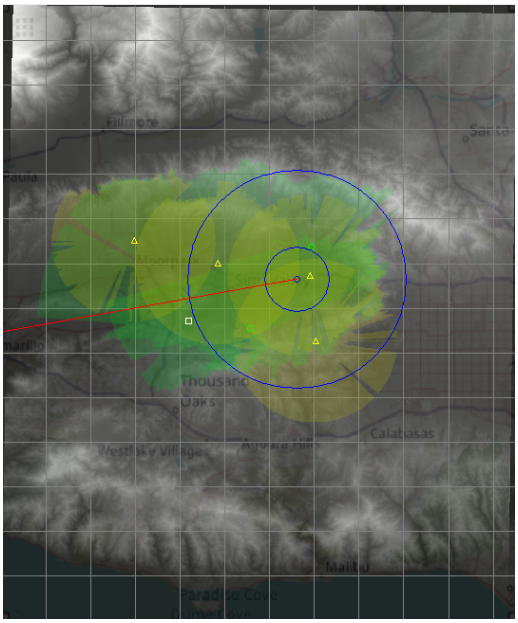


Figure 5.17: Deployment using the defense tactical choice on scenario D with a larger engagement zone than usual. Three of the fire units have formed a triangle around the protection object, maximising their coverage in the engagement zone.

5.3.5 Convergence and execution times

When creating the optimization method the time it would take was of a large concern, as one wants this type of application to be very fast so it can be used in field. The aim was to keep the optimization part of the algorithm around 15 minutes. The execution time of the current application vary depending on parameters and what data is used, but most optimizations that were run when creating these results have been within the time limit.
For the standard scenarios A and D the average execution time for 1000 generations on the large set of points was 13.60 minutes, based on 50 runs. For the small set of points the average execution time on the same scenario was 13.32 minutes. This is not a very large difference, which was unexpected as the number of possible points was thought to affect the execution time. However, taking the number of generations needed for convergence into account, the time for performing an optimization on the larger set would probably take more time.
The number of units that are deployed also affects the execution time, as well as what units are deployed. For example, the alternative deployment of two UndE23, two EldE97 and four EldE98 presented in Section 5.3.3 takes around 10 minutes for 1000 generations.

5.3.6 Communication analysis

To determine how well the simplified communication criteria used during optimization performed, they are compared to a communication application provided by Saab Surveillance. The communication matrix obtained from a typical run on scenario D can be seen in (5.1), and the corresponding deployment is presented in Figure 5.18. This matrix represents which units can communicate with each other, marked with a one, where the first two rows are the UndE23 units, the third is the C2 unit and the last four are EldE97 units. This communication matrix was obtained by calculating lines of sight with a maximum length of 10 km, based on each unit having antennas of height 20 m. For example one can see that the C2 unit is within reach of one of the radar units and all but one of the fire units.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

Figure 5.19 shows the radio coverage from the C2 unit mentioned above, with the other units also marked on the map. This was obtained using a radio coverage calculator provided by Saab, and represents the surface propagation based on several factors such as frequency and antenna heights. The image shows that all units have acceptable radio communication with the C2 unit. However, it is worth noting that the possible coverage extends further using this tool than in the approximation used during the optimization, as that value was reduced to fit the map. When measuring

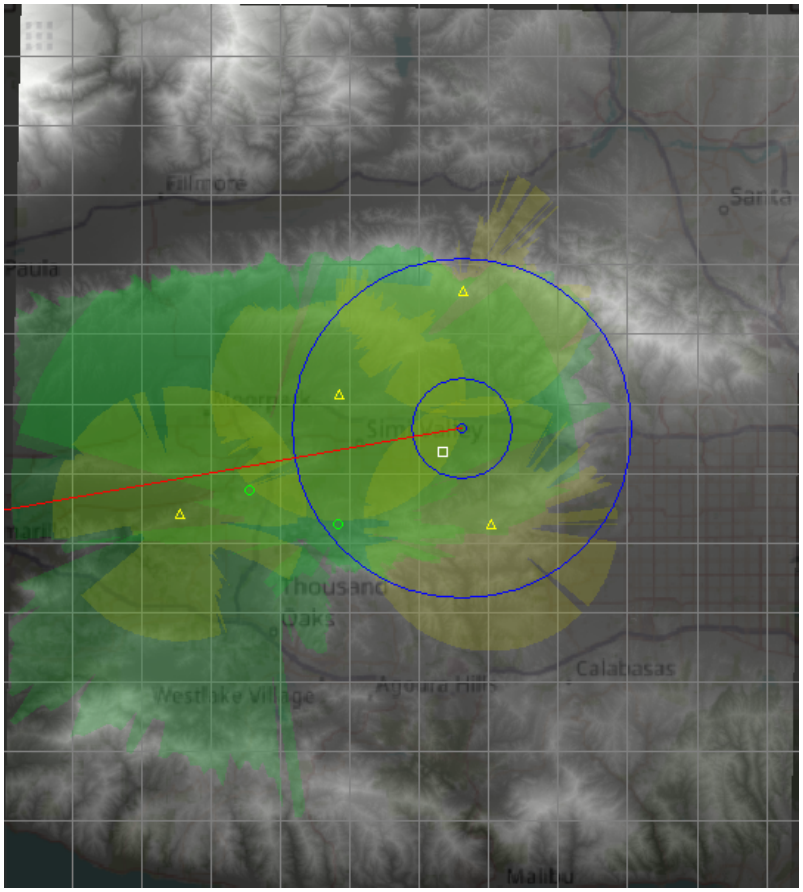


Figure 5.18: A deployment found for scenario D in order to illustrate the communication model. It does not prioritize stable communication.

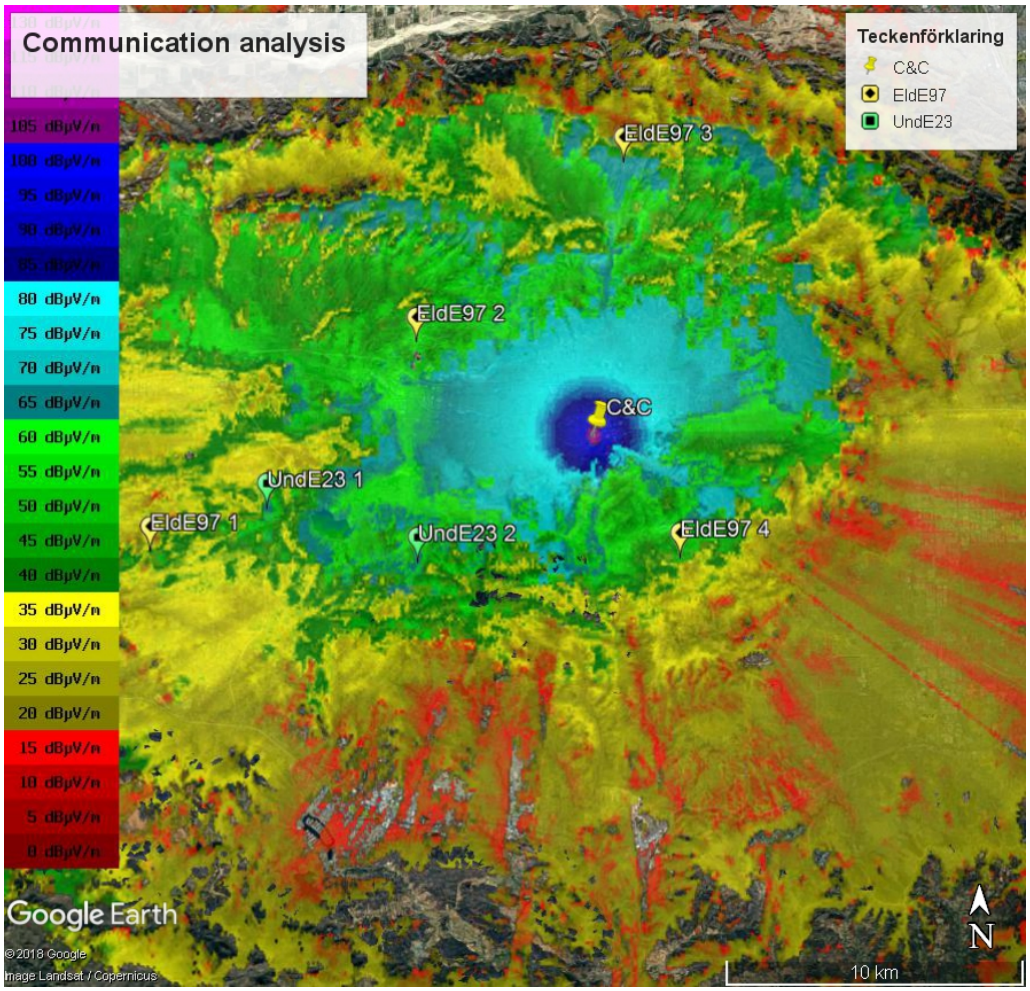


Figure 5.19: Image representing the radio coverage from the C2 unit in a deployment. The image shows how good the signal is expected to be around the unit given the antenna height of the units. In this case the antenna is assumed to be 20 meters tall. One can see that all units seem to have a decent signal from the C2 unit. The image comes from Google Earth Pro, with an overlay obtained using a radio coverage calculator provided by Saab.

the distances between units it becomes clear that one of the fire units and one of the radar units are placed more than 10 km from the C2 unit. This is probably the reason why they are marked as non visible in the communication matrix, and is therefore a consequence of the choices made regarding how to approximate communication. Additional images showing the radio coverage for the other units are shown in Figures B.7 through B.12 in Appendix B.4. These images show that all the units that could be communicated with according to the communication matrix also can be seen using the radio coverage calculator. The calculator shows that some more units should be within reach as well, which is due to it not using the same limitation on the distance as during the optimization.
In Figure 5.20 a deployment run on scenario D which prioritized stable communication is presented. In (5.2) the corresponding communication matrix is found. Note that the deployment has several shortcomings in terms of cooperation and overlap

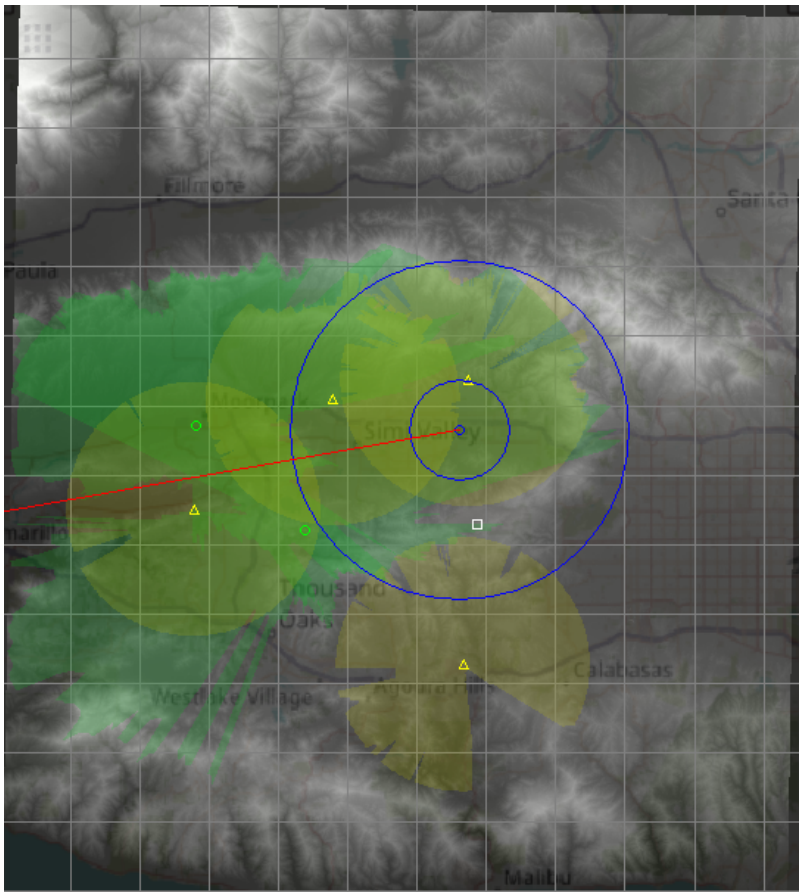


Figure 5.20: A deployment found for scenario D which prioritizes communication. It has some gaps, probably originating from the addition of stable communication into the optimization.

of coverage between the radar units and fire units. Also note that the matrix has an unstable node on row 6 as it is only connected to itself and one other unit. This could mean that the weight for this particular fitness goal can be increased.

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

6
Discussion

This chapter contains discussions regarding different aspects of the project. The methods regarding the usage of the high-resolution data, the preprocessing and the optimization are discussed in detail. In addition, the results presented in Chapter 5 are discussed as well as some future possibilities of the project.

6.1 Data usage

No one can argue that the data provided by Vricon is both more precise and more flexible than the data used in today's military applications. Consisting of both a digital surface map, digital elevation map and corresponding classification data, it can be used in many ways. With a resolution of 0.5 meters per pixel it is also much more detailed than today's standards, allowing for very exact calculations. This thesis explored some potential uses of the data, and based on the results one can draw some conclusions regarding the usage of the data.

6.1.1 Using the Vricon data

One of the benefits of using elevation data from Vricon is that the user can choose between elevation and surface data. The difference lies in that surface data takes the objects on the ground into account, such as trees and houses, whereas elevation data measures the elevation of the ground. It is therefore reasonable to use the surface elevation data for projects where line of sight is of importance, such as this one. Naturally, this is what have been done in all parts of this project.
However, one can in fact argue that the optimal method would be to use a combination of both surface and elevation data for different parts of the algorithm. For example, it would probably be more accurate to use the elevation data when computing the slope of the ground as to evaluate if it is possible to deploy on a site. In our application, this part of the preprocessing is performed using the surface data, which may cause areas to be marked as undeployable due to a steep slope, when in fact the underlying terrain was relatively flat. In the same way, the total height of the antennas computed as $h_{total} = h_{DSM} + h_{antenna}$, may have been exaggerated by the application due to being obtained based on surface data instead of elevation data. A disadvantage of using both surface and elevation data is that one would need to load and store double the amount of data during preprocessing, which would be both memory and time consuming.
As for the classification data it gives the interesting possibility of discarding positions

based on the terrain with almost no effort. In the current data, Vricon classifies terrain with vegetation into three different categories based on the height. One may ask how precise this classification is, and whether it is reasonable to assume that low vegetation, of for example the height 0.5 to 2 meters, can be cut down easily. In a critical situation it would also be important to have data that is up to date, as vegetation changes during the course of the year.
During the course of the project it has also been found that the classification regarding paved surfaces is not optimal, with many miss-classified pixels; sometimes in the middle of nowhere. Roads that are not paved are also often left out of the classification, and instead often labeled as ground. For a mission planning application where the road network is of large importance one should therefore not solely rely on the provided classification data, and instead combine it with similar data from another source.
In future iterations of the project it would be interesting to improve on the data that is used during the preprocessing. An easy step is to use elevation data for the calculation of slopes. As mentioned it would also be interesting to combine the classification of roads with an actual road network, and to evaluate what other data sources one can use. For example it would be of great interest to have data on the bearing in the ground, to evaluate how reasonable it is for a heavy unit to deploy at a position.

6.1.2 Downsampling and using less points at a distance

A solution for the storage and memory issues that arose due to the enormous amount of data is to use different resolutions at different distances. The best way to obtain data of lower resolution is to use downsampling, creating a representation of the same data but with fewer pixels. The time consumption of different resolutions of data are compared in Table 5.1 and 5.2 in Section 5.1. Based on these numbers it appears that there is not a significant time difference in downsampling to different sizes of resolution. Performing line of sight calculations of a fixed stepsize is not affected much by the resolution of the data, as long as it all can be contained in a single matrix. Using a stepsize equal to the resolution causes an increase in running times for higher resolutions. This should not come as a surprise, as one performs the computations for a larger amount of points in that case.
Based on this it is reasonable to conclude that one should aim to use as high resolution as possible in the downsampling, while still being able to store it in the random access memory without problems. The maximum resolution will therefore vary depending on the size of the area that is used, as well as on the computer. However, based on the file sizes obtained for the data used in this implementation it seems that one should be able to use a resolution of 10 meters without any issues. The implementation in this thesis used 10 meters, but 5 meters could probably have been used as well without affecting computation times in a significant manner.
A risk that comes from using lower resolution of data compared with the original DSM data is the risk of missing important large items, obscuring the sight at a deployment site. For example, a tree that is positioned a hundred meters from a unit may block a few degrees of the bearing. How much such an object affects the general

view is largely dependent on the distance to it. Using some basic trigonometry one can calculate that a change in angle of 1°corresponds to a movement along the arc of the circle larger than 30 meters at distances above 1.7 km, and a movement larger than 10 meters in the arc at distances above 570 meters. Depending on how precise the evaluation of bearings is, one can therefore start using lower resolution data at a quite close range. During the course of this project the lower resolution data of 10 meters was used at distances above 4 km, which should be well above the limit of where it would matter.

6.2 Preprocessing methods

The traditional way of representing the sight of a unit is to use viewsheds, which were presented in Section 2.4. However, viewsheds are both time and memory consuming, with the fastest being R2, running in $\mathcal{O}(n_R^2)$ time where n_R is the average density of points for the visible radius of the observer. Usually the results from a viewshed are saved as a matrix, which needs to be large enough to contain all the pixels that can be seen from the observer. This matrix would be of a size $n_{max} \times n_{max}$, where n_{max} is the point density for the maximum radius. Instead two other measures, referred to as sight polygons at an elevation and at a specified height, were created. These measures will be discussed in detail below.

6.2.1 Sight polygons at an elevation

The first measure consists of evaluating the distance that can be seen for a given bearing and elevation. This is performed for k bearings and l elevations, and the distances are recorded in a matrix of size $k \times l$. The areas of these cone shaped surfaces is also saved and summarized. This measure has sometimes been referred to as a volume of coverage, which is not completely true. However it is still a measure that becomes much smaller if there are objects in the way at low elevation angles, and it is quite computationally and memory efficient. In a sense it is also very similar to how a radar unit "sees". Still, it might be of interest to try to model the volume of coverage in another way, or to evaluate how accurate this measurement of sight is.

6.2.2 Sight polygons at a height

The easiest way to represent a combined coverage of several units is to study their coverage at a certain height above ground or a certain height above sea level. In this project we chose the height above ground, due to the large fluctuations in ground height in the Los Angeles data set.
The sight polygons at a given height are in a way very similar to viewsheds, evaluating a large number of points at a certain height. However, instead of performing these calculations for every point in range it was decided to create a lower approximation of the sight, corresponding to the uninterrupted area which can be seen. To avoid storing large matrices of data the evaluation is performed at regularly spaced bearing angles. This way one only needs to store a vector of the same size as the

amount of angles that are evaluated, containing the corresponding distances that can be seen.
While these polygons are not as exact as viewsheds, they still are able to give a lower estimate of the coverage. At heights above ground that are of interest for military operations, this lower estimate is also likely to be quite similar to what a viewshed would produce, as the terrain would need to change rather quickly for the sight to be obscured temporarily.

6.2.3 Identifying the best points

The method of finding points contains many steps and simplifications. As there are about 10^9 points considered possible to deploy on in the Los Angeles data set, it would take too much time to evaluate them all even when using approximate measures. Additionally many of them are in close proximity of each other causing there to not be much of a difference between them. Therefore the choice was made to reduce the amount of points evaluated to one every s meter, with s being a parameter for the user. A subset of the points in a square of size $s \times s$ were therefore evaluated using the approximate measure VIX, presented in Section 2.4.2, and the best point was chosen as a representative of that square.
A risk with this approach is that good points are lost because of performing badly on the approximate measure. This is a valid concern, and one may consider increasing the number of samples that VIX is based on. Also, there may exist points in the square that are not evaluated at all but are excellent positions for a unit to deploy on. However, due to time constrictions there is no possibility of evaluating every single position. Once the units are deployed in the field they of course have the possibility of changing position to a different more optimal one.

6.3 Simplifications

The largest simplification of this project is the usage of a smaller map than what would be used in a real situation. The data set over Los Angeles covers an area of approximately 50×50 km, whereas in a real application the side-length of the area would be larger than 200 km. Based on this all the distances were reduced to fit inside of the area used in the application. For example, the maximum reach of a UndE23 was set to 20 km, and the radius of an EldE98 was set to 2.5 km. The relative differences were kept similar to reality, with the exception of the communication distance. While distances related to sight can be reduced without causing any problems, reducing the maximum distance for communication will affect the possibility of an operational deployment greatly. The terrain is also not modified to account for using shorter distances, creating problems for communication. Therefore the default maximum distance for communication was only set to 10 km, whereas the real distance is somewhere close to 20 km.
One of the main simplifications that have been made regarding the preprocessing is how the accessibility to a position is computed. Ideally, the locations should be easily accessed from a road, without needing to cross any steep slopes or high vegetation. The ground should also be able to carry a heavy truck of several tonnes. However, in

this application most of these were ignored, and instead a simplified version based solely on distance from the road was implemented. A reason for this is that the project already had many other complex and complicated aspects to consider. This could however be implemented in a later version during the preprocessing of the data.
There are also several limitations regarding the modelling of the area that can be viewed from a unit. In reality a radar has the possibility to see objects that are not in its line of sight due to diffraction and reflection. This was ignored during the project as to keep the calculations relatively simple. Regarding communication, which also was modeled using line of sight, there are even more aspects that may affect the result, ranging from weather to density of the vegetation. All of these were ignored. However, in Section 5.3.6 a comparison between the simplified model and a more accurate model is made.

6.3.1 Military aspects of the project

The application considered in this project had a rather limited choice in possible missions and objectives. This was a conscious decision, as the project contained a large variety of tasks. As we started the project with a blank slate of knowledge about the military aspects, it was also hard to determine what parts were more important than others. The choice was made to use one standard scenario that could be changed slightly based on user inputs. This scenario represents one of many situations in which one may want to use a mission planner.
This project only considered a small selection of the units of interest for an air defense company, which could consists of a completely different composition all together. One such unit is the sensor unit ARTHUR. This unit models the radar horizon, determining at which height and elevation angle the horizon begins. While it has not been implemented during the course of this project, it is certainly possible using methods similar to those described in Section 4.3.3.
In addition, the fire unit EldE98 was simplified during the project to only consider its ability to lock on the target before launch. In reality however, it also has the ability to Lock-On-After-Launch. This would let it surpass some of the restrictions regarding the height of the nearby terrain in regards to coverage. However, the unit would still need to be within communication range with the other parts of the company, thus this advantage would not be as impactful.
As one can expect that the miliary operates in a very practical manner some of the limitations would in reality not be as important. For example, when deciding between possible deployment sites during preprocessing there was a strict restriction on there not being a single object exceeding the unit's height for a certain radius around the site. In reality however, if a site is deemed good enough to deploy on but a single tree would block the view, then it would probably be cut down.

6.4 Optimization

The result of the project is an application that successfully manages to deploy a unit based on a scenario, with results that make sense from a tactical view. However

there are still plenty of aspects to discuss, ranging from the details in the results to the objective function, which is done in the following sections.

6.4.1 Analysis of results

The majority of the results obtained in Section 5.3 from the optimization were expected. The typical deployments found in Section 5.3.1 solved the deployment problem with an intuitive choice of sites to deploy on. However, for scenarios where the possible points obtained from the preprocessing were not centered around the KOZ the optimization had a difficult time to find a satisfying solution. This was mainly the case for scenario B and scenario C. It is also worth to keep in mind that the area to deploy on is not ideal for many units, as the terrain is mountainous. Thus, a radar unit will not be able cover its entire potential coverage area as the mountains and hills obscures its sight.
The addition of optional partial objectives to the optimization, such as having double coverage in the EZ in the PTL direction yielded satisfying results, as the algorithm tented to favor a deployment of fire units which satisfied this. Examples of this could be found in Figures 5.6 through 5.9.

6.4.1.1 Using different areas containing deployment points

In Section 5.3.2 the optimization method was tested on different sets of deployment points. The original set of points is also called the larger set, and contains about 9000 possible deployment points for each unit, spread out on an area of 80 maptiles. A smaller set containing about 4000 locations, which are spread out on 36 maptiles was also created. In general it seemed as if the smaller set tended to find better solutions more similar to each other. However, in Table 5.5 one could see that the mean value of the highest fitness was lower compared to using the larger map for both scenario A and D. A possible reason for this is that the larger map can find positions with good coverage in areas that are not that relevant for the current optimization, resulting in a higher fitness but a less suitable deployment. When using a more limited set of points, located in relevant locations, this can be avoided. Another reason could be that the optimization on the larger map did not have the time to converge in a satisfying manner, or that the values of the weights α_i are not optimally balanced.
Another result regarding the same subject is that the convergence of the smaller set of points is faster than the convergence of the larger set, which was tested by recording the number of generations needed to reach a certain fitness. The histograms in Figure 5.11 clearly indicate that the results using the smaller set of points have either a faster convergence, or in general better results. There are probably several reasons for this, such as the amount of possible points used and that the points are more spread out in the larger set. However, it is clear that in order to obtain an efficient optimization one should choose the areas in which one wants to deploy carefully. A larger number of deployment points, in positions not of interest, will cause the optimization to run slower, the convergence to take more generations and the creation of the points to be more time consuming than necessary.

6.4.1.2 Differences based on tactics

In order to investigate how the algorithm deploy the units for different missions, the additions of choosing different tactics were added to the optimization. These resulted in a choice of different values for the weights α_i's first presented in Chapter 3.
When deploying in a defensive manner the objective of the formation is to cover as much of the EZ as possible in order to secure the KOZ. Presented in Figure 5.14, the algorithm tended to deploy the units to cover as much of the EZ as possible with the fourth unit placed further away towards the PTL, which also is an intuitive solution. However, as the KOZ in the default scenario D had the radius of equal size as the range of the fire unit, it tended towards choosing solutions utilizing only two units to cover the EZ. A larger radius for the EZ results in the fire units being placed in the more intuitive way of a triangle in order to optimally cover the KOZ, as depicted in Figure 5.17.
The choice of optimizing for surveillance had the intention of covering as large portion of the map as possible, with the risk of having worse coverage inside the EZ. In Figure 5.15, this was the case as the units were much more spread out and covering a larger area compared to when optimizing using a defensive prioritization. When using the default mode, the optimization should try to find a deployment with both the objectives of good surveillance and defense in mind. This was found to be the case as is shown in Figure 5.13, where the deployment satisfies a combination of both objectives.
The tactical choices also had another interesting result. As was presented in Figure 5.16, many of the solutions found by the algorithm were more or less feasible, especially early on. The choice in the weights α_i through the different tactics lets the algorithm know what to prioritize when trying to find feasible solutions. For instance, the very strict communication requirement, described by the penalty P_1 in Chapter 3, makes all solutions unfeasible if they were incapable of communication. Thus having a tactical choice of defense, were the algorithm prioritizes to keep the units in a closer and tighter formation, lets the algorithm easier fulfill this strict requirement. In comparison, the choice of prioritizing surveillance tells the algorithm to spread out the deployment, which results in it having a more difficult time to find a formation capable of communication, especially given the mountainous terrain. One could further investigate how to choose these weights in order help the algorithm converge faster.
As the composition of an air defense company can vary greatly a formation of a different composition of fire units were investigated. The results of this was presented in Figure 5.12. As the amount of EldE97 units were drastically decreased, resulting in a smaller possible area of coverage. The algorithm tried to work its way around this limitation by favouring deployments where the EldE97 had a very good all around coverage area. As the EldE98 units has a much smaller area in which the enemy can be engaged, many of the possible deployment sites for these units had a very good coverage area compared to EldE97. Thus, the algorithm instead chose to patch together the holes in the coverage of the EZ with the help of these units. A more intuitive way of utilizing the EldE98 units would be to let them cover the KOZ and placing the EldE97 further away to better cover the PTL.

6.4.1.3 Convergence

This subject has already been touched upon a few times, but it is one of the most important aspects regarding the optimization algorithm. It is clear that the area used during the optimization is difficult to deploy on, with several mountains surrounding the cities. There are therefore a great deal of possible solutions that are equally good. This causes the optimization to not converge to a global maximum, if there even exists one. Instead it explores different strains of solutions tending to have similar sizes of coverage and be deployed in a similar manner. It is possible that it would be easier to find a more general optimal solution on a more flat map, but it is hard to say in advance without testing.
The convergence takes longer time the more complex the problem is, which was shown earlier regarding the smaller and larger sets of points. It is likely that the same applies to adding more units or creating a more complicated objective function. Therefore it is hard to say an optimal number of generations that should be used. This needs to be decided by the user, taking into account both the complexity of the problem and how much time it is allowed to take. However, based on the results in this report, 1000 generations seem suitable for the smaller list of points and the standard scenario. On the large set of points one could probably have benefited from running the algorithm for more generations, perhaps 1500 or 2000, which would increase the execution time to around 30 minutes.

6.4.2 The objective function

The objective function had a great deal of different parameters to consider as it had to model and evaluate how good a deployment site was for every unit, as well as the different combinations of them. It also had to model the scenario and how the current deployment would perform on it.
The genetic algorithm optimizes based on a fitness, and the better the fitness the better the solution. Thus, the choice fell on modelling all the different aspects of the problem individually, then summarizing these to obtain a total fitness. This way, every fitness term describes how well a current solution performs on the specific aspect of the problem, in which the fitness corresponds to. As the problem is very complex, having a deployment which performs well on one partial objective could as a result of this perform worse on another part. Thus, the solutions obtained from the optimization are similar to being Pareto optimal, meaning that no solution is the best for every variable at the same time.
Some of the fitness terms aims to optimize the individual units performance, while others evaluate the deployment based on some important tactical aspect typical for a deployment or one partial objective of the scenario. Thus, the combination of the fitness terms should describe how well the deployment performs on all of these aspects combined. As it was difficult to tell what aspect was more important than another, and especially to determine which partial fitness term contributes more to a good solution in general than others, they were all normalized to the same interval and multiplied by a constant. This constant would then give the option for the user to let the algorithm know what to prioritize during the optimization process. Thus, if the optimization yields strange results, this could be a underlying factor.

However, some of the partial fitness terms could also be modeled in a better way, resulting in a more satisfying optimization process.
As the fitness aims to optimize many different aspects at once, the choice of using penalties was done to let the algorithm clearly know whether it should discard a solution or try to explore it. In addition, some of the penalty factors were included to model tactical aspects which did not quite fit as a fitness. An example of this is the first penalty P_1 which lets the algorithm know if all units are able to communicate using the same network of communication. As all units have to be able to communicate with each other at all times, all solutions where this is not fulfilled should immediately be discarded in favor of solutions able to communicate using the same network. Thus if it was a fitness it should be increasing if the solution was doing well in this aspect, but as it was only a restriction of the ability of the deployment it was more suitable to model it as a penalty.
One aspect which was not explicitly modeled was to let the algorithm know if it should prioritize to place units between the PTL direction and the object to protect. This could be done as a penalty, forcing the algorithm to have a fixed number of units in between. It could also be modeled as a fitness, possibly defining this space in between as an area and try to maximize the coverage of this area by the relevant units. This is similar to how the EZ coverage was implemented, which gave satisfying results.

6.4.3 The choice of using a genetic algorithm

One of the reasons for choosing a genetic algorithm for the optimization was the fact that the objective function for the problem was very hard to define. The objective function has many factors to consider in order to obtain a good and working deployment of an air defense company. Thus, trying to model a differentiable objective function, which is a strict requirement for many classical methods of linear and integer optimization, would have taken a lot of time better spent somewhere else and might have not even given something of use in return. For this reason, the gradient descent and stochastic gradient descent methods were discarded early on in the project. As the search space is not continuous, more resembling islands where the search tends to jump between, other methods requiring a continuous search space were discarded. An example of this is the particle swarm optimization shortly introduced in Section 2.7.2.2. It is in general a good choice of method to solve a problem of a similar nature with many local optima, but the discontinuity makes it hard to make good use of the the particles' velocities. Hence, this method was discarded as well.
Another strength of the genetic algorithm is that it can explore many solutions at a time depending on the genetic operators, and how one chooses to apply crossover to the chromosomes. A higher mutation rate could let it explore much more of the search space, or a different population size could give it a wider search space to start with, at the cost of calculation speed. The choice of using varying mutation rate was based on the nature of the points to optimize. As these points are not described as a continuous function, the algorithm could stop exploring new genes and instead have less variation in the population than what was wanted. Thus, an increase of

mutation rate helps with forcing it to explore more of the search space after it has spent too much time exploiting a locally optimal gene.
A disadvantage of a genetic algorithm is that while it finds an optimum, it is not necessary the global one. However, this holds for a lot of other optimization methods as well, especially for non convex problems. A property of the algorithm is that if it finds a good gene it will try to exploit it. This however, also holds for bad genes and it might spend an unnecessary amount of time exploring a non optimal solution as it is the best one it has found so far.
The decision on what values to use for the genetic operators such as population size and tournament probability, tends to differ from problem to problem. Because of this, altering a few aspects such as factors and terms in the objective function could change what values should be used by the algorithm. Thus, to find the optimal values for genetic operators one would need to investigate these values for a great variety of problems. It is probable that there are parameter values which yields a better result than what has been presented here, as a through investigation has not been performed in this thesis.
An operation which was considered for the algorithm was to have populations on separate islands isolated from each other, and during a later stage merge these together. This could potentially help with the fact that the genes could tend to get stuck exploiting one gene in the population, thus finding the best combination of these locally optimal solutions after the merge. This is not guaranteed and it would also add to the computation time of the algorithm which was already on the edge of what the optimization was allowed to take.

6.5 Future works

It seems as if it is possible to create a functional mission planning tool making use of the high resolution data from Vricon, even if some of the details still needs to be investigated further. Some examples of how the current application could be improved upon are discussed in the following sections.

6.5.1 Preprocessing

Some improvements regarding the preprocessing have already been mentioned, with the most important being a more detailed evaluation regarding what positions are accessible for the units. The addition of more data would also be beneficial, for example performing parts of the computations using elevation data instead of surface data, and combining it with a more detailed road map. All of these are likely not that difficult to implement, and would not affect the computation times significantly.
It could also be interesting to give each position a value representing how suitable it is be to deploy on, based on factors such as its slope and the distance to a road. This value could then be evaluated in the objective function during the optimization, allowing for choosing less optimal positions if necessary.
Another addition that could be of interest is to add a step of identifying and more accurately reconstruct the heights of buildings, as was done in the thesis by Mats

Nilsson at Linköping University [19]. This would be of importance when considering to deploy in close proximity to an urban area.
Finally, while the measures of the sight of a unit that have been created seem to be accurate there is likely other methods that also could be useful. One could also implement a varying step size in the methods for calculating line of sight, allowing for quicker evaluations of long distances.

6.5.2 Optimization

As some of the results of the optimization could be argued to have been influenced by some of the restrictions, such as a difficult area to deploy on, it would be of interest to be investigated further. For instance the application could be tested on another more forgiving area, where the units have an easier time finding deployment sites with a good coverage area. This is something that probably should be done in order to thoroughly evaluate the potential of the method.
Another thing to investigate would be to have varying weights on the different partial fitnesses in the objective function. As argumented earlier, the results presented in Section 5.3.4 could indicate that promoting a close formation for the deployment during an earlier stage of the optimization could result in finding a feasible solution at an earlier stage. This would in turn give the algorithm more time to explore feasible solutions as it would find a feasible candidate solution much earlier on. If one would want to have a deployment more focused on surveillance of the area, then the corresponding weights of the fitness function could be increased after some number of iterations.
The parameters of the genetic algorithm could always be improved in order for it converge more quickly. One could however also investigate how a hybrid version of the genetic algorithm would perform of the problem. As an example, one could let the genetic algorithm first find an optimal solution. This solution could then be given to another algorithm as a starting point. Then a new optimum would be found if the other algorithm would improve the solution it started with. Another alteration of the genetic algorithm would be one which has populations on separate islands. These would then be merged at later point of the optimization. This would let the algorithm first explore local optimal genes of the population, and then combine these locally optimal genes in order to find a combination which improves the solution.
One could also further explore what is modeled in the objective function and add, remove, or improve terms already described. An example of this could be to add an area which would need coverage in the direction of the PTL, but outside the current EZ for both radar and fire units. This could help guide the algorithm to place units closer to the more threatening direction of PTL for the given scenario. However, one needs to keep in mind that this is likely to affect the execution time of the algorithm.

6.5.3 Other possibilities

As the main focus of the project has been to find static deployments, it would be of interest to investigate how well the solutions would perform in a sequence,

as it would be used in a critical situation. An example of this is the deployment scheduling which was investigated by Kevin Larsson for an ARTHUR unit [18]. This would test how easy it would be for a subset of the most promising deployments found by the genetic algorithm to rotate between deployments in a schedule.
It would also be of interest to see if it is possible to narrow down the final solutions using dynamic test scenarios, imitating how the units would perform in a battle. While the use of dynamic test cases has been deemed to time consuming to use during the optimization, it could still be used as a final evaluation when deciding between possible deployments.

7
Conclusion

During the course of this master's thesis it has been shown that it is possible to create a mission planning application for the deployment of air defense companies based on high-resolution geospatial data. The resulting deployment solutions indicate that the model is a realistic approximation of the real world scenarios encountered during military deployment planning. However, the application needs to be tested further on different data sets and terrains, as only one data set has been used during the implementation.
A large part of the thesis has been to investigate the usage of high-resolution geospatial data provided by Vricon. The data has been found to contribute with valuable information for a realistic application. However, one needs to consider carefully when such high-resolution data is actually beneficial as it increases memory usage and execution times greatly.
In conclusion there is a great potential in this type of military planning tool, especially when making use of high-resolution geospatial data. While this application was created as a proof-of-concept, one can surely create an application satisfying the needs of a military planning tool by improving upon the methods used and performing further tests on different data sets.

Bibliography

[1] Gary Chartrand. *Introductory Graph Theory*. New York: Dover, 1984.

[2] B LOUIS Decker. *World geodetic system 1984*. Tech. rep. Defense Mapping Agency Aerospace Center St Louis Afs Mo, 1986.

[3] Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. "Image analysis using mathematical morphology". In: *IEEE transactions on pattern analysis and machine intelligence* 4 (1987), pp. 532–550.

[4] John W Hager, James F Behensky, and Brad W Drew. *The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS). Edition 1*. Tech. rep. Defence mapping agency hydrographic/topographic center, Washington DC, 1989.

[5] John W Hager et al. *Datums, ellipsoids, grids, and grid reference systems*. Tech. rep. Defence mapping agency hydrographic/topographic center, Washington DC, 1992.

[6] John H Holland. "Genetic algorithms". In: *Scientific american* 267.1 (1992), pp. 66–73.

[7] Wm Randolph Franklin, Clark K Ray, and Shashank Mehta. "Geometric algorithms for siting of air defense missile batteries". In: *Battelle Inc., Columbus OH* (1994).

[8] Darrell Whitley. "A genetic algorithm tutorial". In: *Statistics and computing* 4.2 (1994), pp. 65–85.

[9] Yuhui Shi et al. "Particle swarm optimization: developments, applications and resources". In: *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*. Vol. 1. IEEE. 2001, pp. 81–86.

[10] FMV. "Försvarets Telenät Systembeskrivning". In: (2003).

[11] E Roy Davies. *Machine vision: theory, algorithms, practicalities*. Elsevier, 2004. Chap. 6.

[12] W Randolph Franklin and Christian Vogt. "Tradeoffs when multiple observer siting on large terrain cells". In: *Progress in spatial data handling*. Springer, 2006, pp. 845–861.

[13] JC Iliffe and Roger Lott. "Datums and map projections for remote sensing". In: *GIS and Surveying, 2nd edition (Dunbeath: Whittles Publishing)* (2008).

[14] Mattias Wahde. *Biologically inspired optimization methods: an introduction*. WIT press, 2008.

[15] Lauren A Hannah. "Stochastic optimization". In: *International Encyclopedia of the Social & Behavioral Sciences* 2 (2015), pp. 473–481.

[16] Fredrik Aas Isaksen and Kim Nilsen Brusevold. “Mission Planning with High-Resolution Geographical Data”. Bachelor’s thesis. Østfold University College, 2018.

[17] Léon Bottou, Frank E Curtis, and Jorge Nocedal. “Optimization methods for large-scale machine learning”. In: *Siam Review* 60.2 (2018), pp. 223–311.

[18] Kevin Larsson. “Deployment Planning for Artillery Hunting Radar Systems Using High-Resolution Geospatial Data”. Master’s thesis. Chalmers University of Technology, 2018.

[19] Mats Nilsson. “Building Reconstruction of Digital Height Models with the Markov Chain Monte Carlo Method”. Master’s thesis. Linköping University, 2018.

[20] National geospatial-intelligence agency. *Digital terrain elevation data, DTED*. `https://www.nga.mil/ProductsServices/TopographicalTerrestrial/Pages/DigitalTerrainElevationData.aspx`. [Online; accessed on 7th Aug. 2019].

[21] Google Earth. `https://www.google.com/earth/(`. [Online; accessed on 7th Aug. 2019].

[22] GeoTools. *About GeoTools*. `http://geotools.org/about.html`. [Online; accessed on 7th Aug. 2019].

[23] LocationTech. *LocationTech Java Topology Suite*. `https://projects.eclipse.org/projects/locationtech.jts`. [Online; accessed on 7th Aug. 2019].

[24] “Metodhandbok Ledning LVbataljon”. In:

[25] OpenCV. *About OpenCV*. `https://opencv.org/about.html(`. [Online; accessed on 7th Aug. 2019].

[26] Vricon. *Company*. `https://www.vricon.com/company/overview/`. [Online; accessed on 11th July 2019].

[27] Vricon. *Vricon DSM - digital surface model with superior accuracy and global coverage*. `https://www.vricon.com/wp-content/uploads/2017/06/Vricon_DSM_print.pdf`. [Online; accessed on 11th July 2019].

A

Parameters

In this section tables containing parameters and values used in the application can be found. Table A.1 contains the parameters and constants that were used for the preprocessing of the map. Table A.2 lists the parameters regarding the units that were used, with the modification for the smaller size of the map. In Table A.3 and Table A.4 one can see the α-values that were used in different miliary tactics as well as the parameters used for the genetic algorithm.

Table A.1: Parameters used for preprocessing the map

Maximum deployable slope	10 °
Maximum distance from road	300 m
Minimum deployment area	300 pixels
Resolution of downsampled map	10 m

Table A.2: Parameters used for the units, with modifications as to fit the smaller map that is used.

Unit	**UndE23**	**EldE97**	**EldE98**
Default number of units	2	4	0
Sensor height	13 m	4 m	4 m
Maximum distance reached	20000 m	8000m	2500 m
Observing height for targets	200 m, 600 m	200 m, 600 m	200 m, 600 m
Elevations	0 to 8°	0 to 8°	0 to 8°
Number of elevations	10	10	10
Number of bearings	180	180	180
Samples in VIX	50	50	200
Samples in a bucket	NA	1000	1000

Table A.3: Values of α that are used for the different tactics.

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8
Normal	1.5	1	1	1.5	1	1	1	1
Defence	1	1	3	1	3	2	1.5	1
Surveillance	4	1	1.5	4	1.5	1	1	1

Table A.4: Standard choices for the parameters used in the genetic algorithm

Number of generations	1000
Population size	20
Tournament size	3
Tournament selection probability	0.7
Crossover probability	0.9
Mutation probability	0.01
Elitism size	3

B

Additional results

B.1 Details regarding standard scenarios

Table B.1 contains the geographical references regarding the standard scenarios that were used for the results.

Scenario	**A**	**B**	**C**	**D**
Latitude	34.259374	34.153058	34.404190	34.278223
Longitude	-118.819942	-118.810406	-118.756309	-118.713425
PTL	90 °	290 °	90 °	260 °
Radius KOZ	3 km	3 km	3 km	3km
Radius EZ	10 km	10 km	10 km	8 km

Table B.1: Parameters describing the standard scenarios used for creating the results.

B.2 Additional results based on the smaller set of points

Results on for the two scenarios A and D on the smaller set of points. The default company set up was used with 2x UndE23, 1x C2 unit and 4x EldE97. The algorithm ran for 1000 generations.

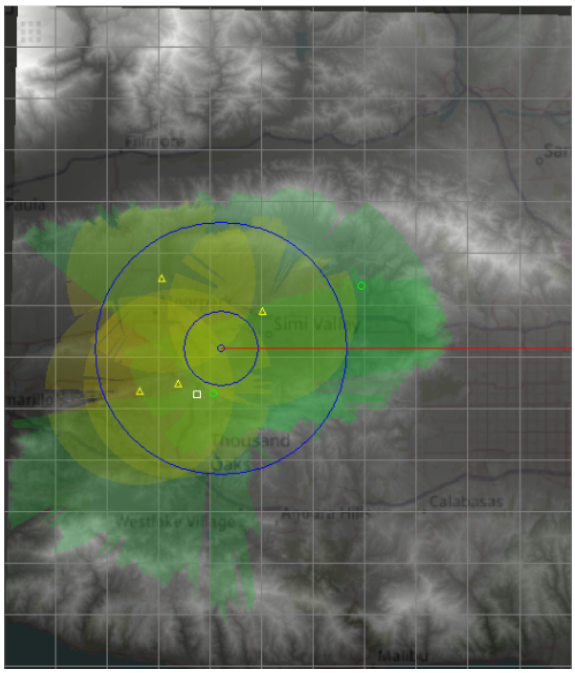


Figure B.1: A resulting deployment for scenario A on the smaller set of possible points, with fitness value 7.167.

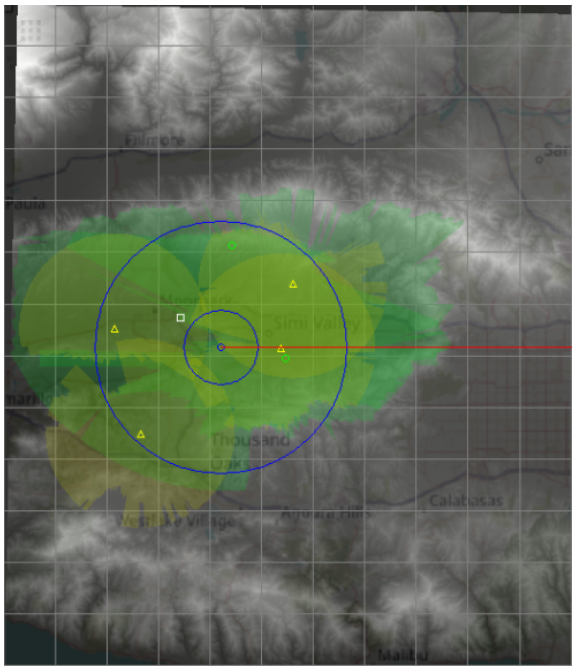


Figure B.2: A resulting deployment for scenario A on the smaller set of possible points, with fitness value 6.738.

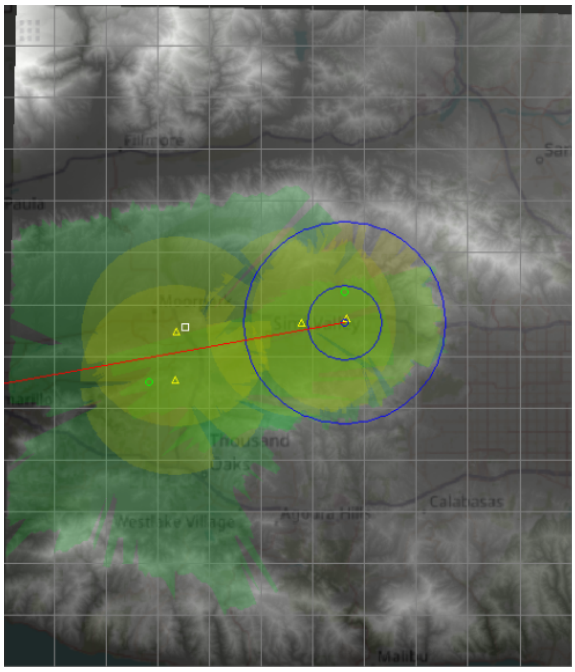


Figure B.3: A resulting deployment for scenario D on the smaller set of possible points, with fitness value 6.574.

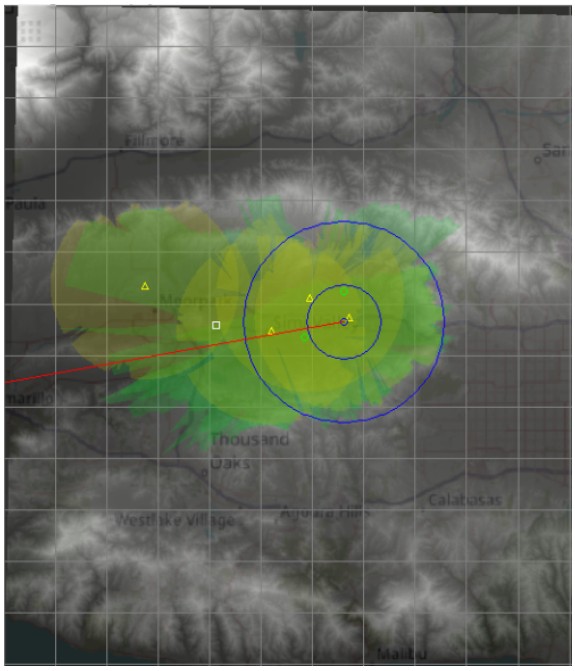


Figure B.4: A resulting deployment for scenario D on the smaller set of possible points, with fitness value 6.203.

B.3 Deployments using two UndE23, two EldE97 and four EldE98

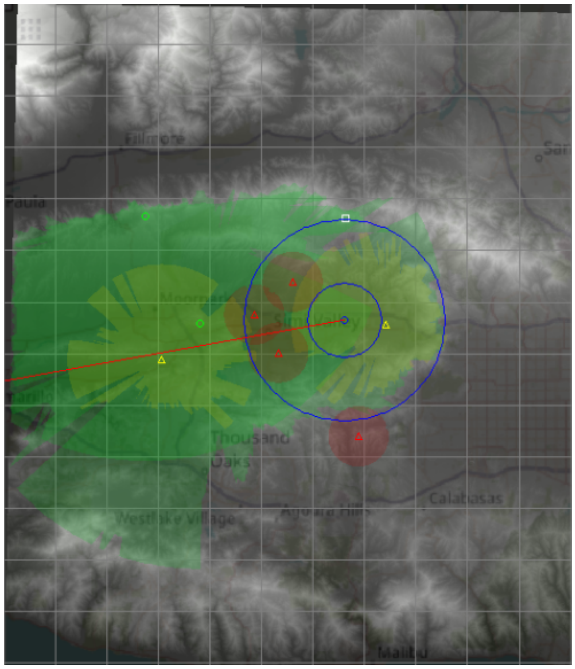


Figure B.5: Deployment using two UndE23, two EldE97 and four EldE98.

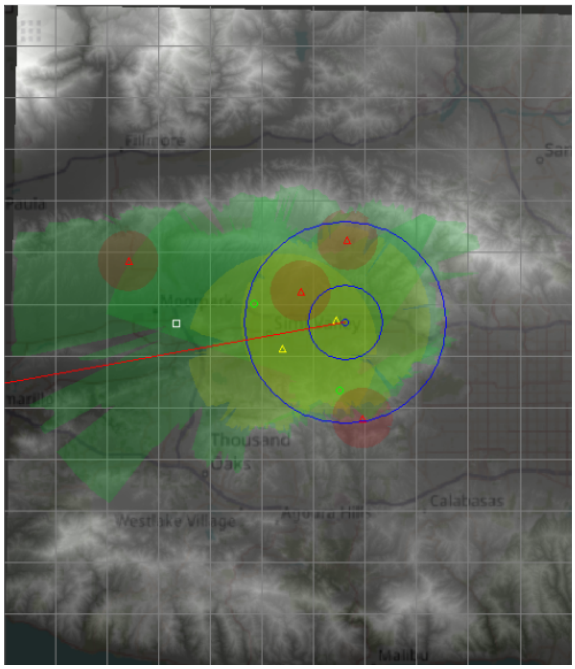


Figure B.6: Deployment using two UndE23, two EldE97 and four EldE98.

B.4 Results regarding communication

These images show radio coverage estimations for a deployment on scenario D, computed using a tool obtained from Saab, which were mentioned in Section 5.3.6.

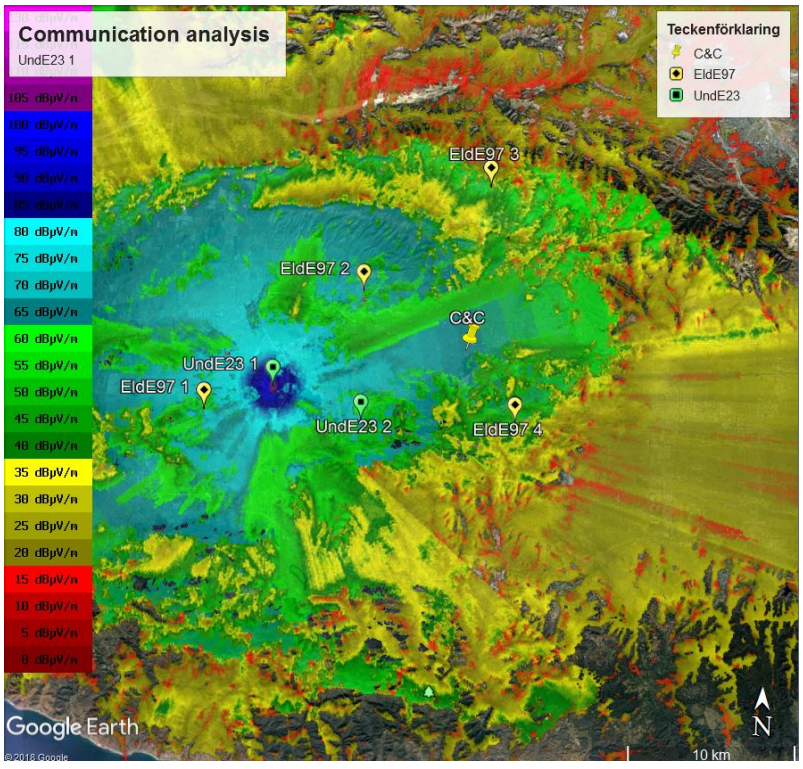


Figure B.7: Radio coverage estimation for the first radar unit.

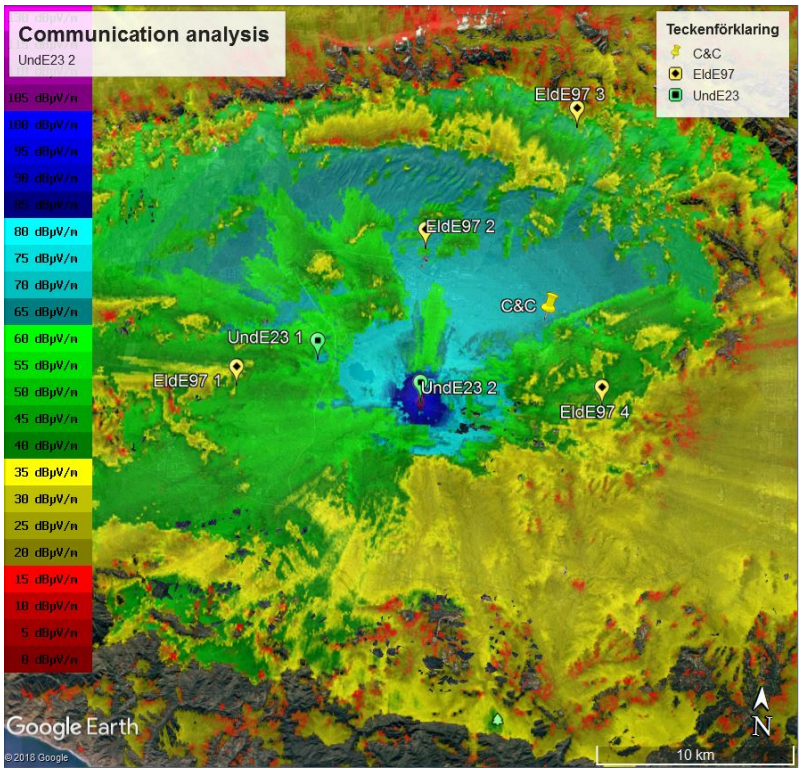


Figure B.8: Radio coverage estimation for the second radar unit.

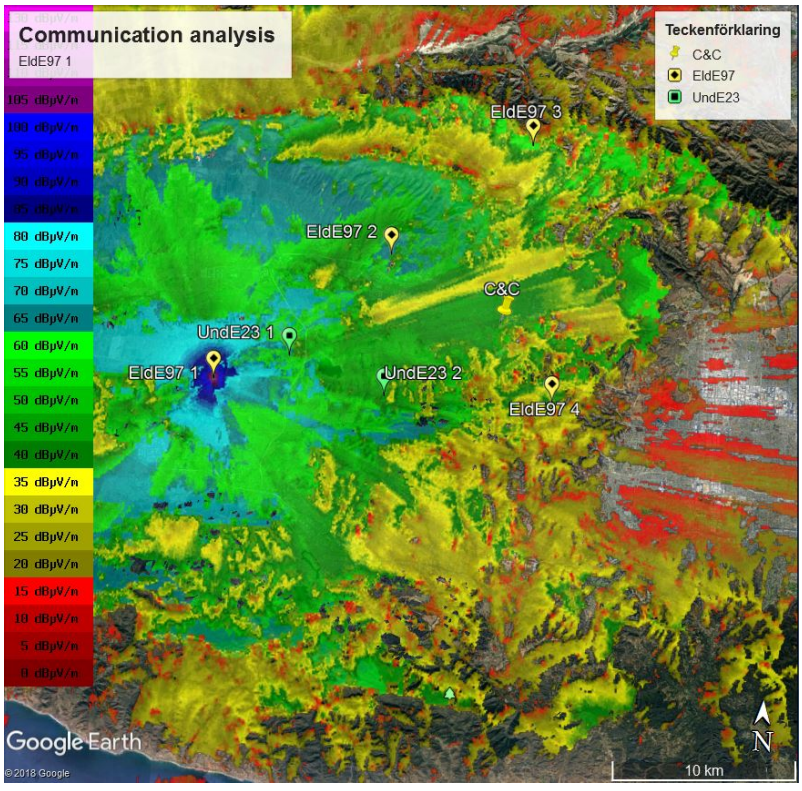


Figure B.9: Radio coverage estimation for the first fire unit.

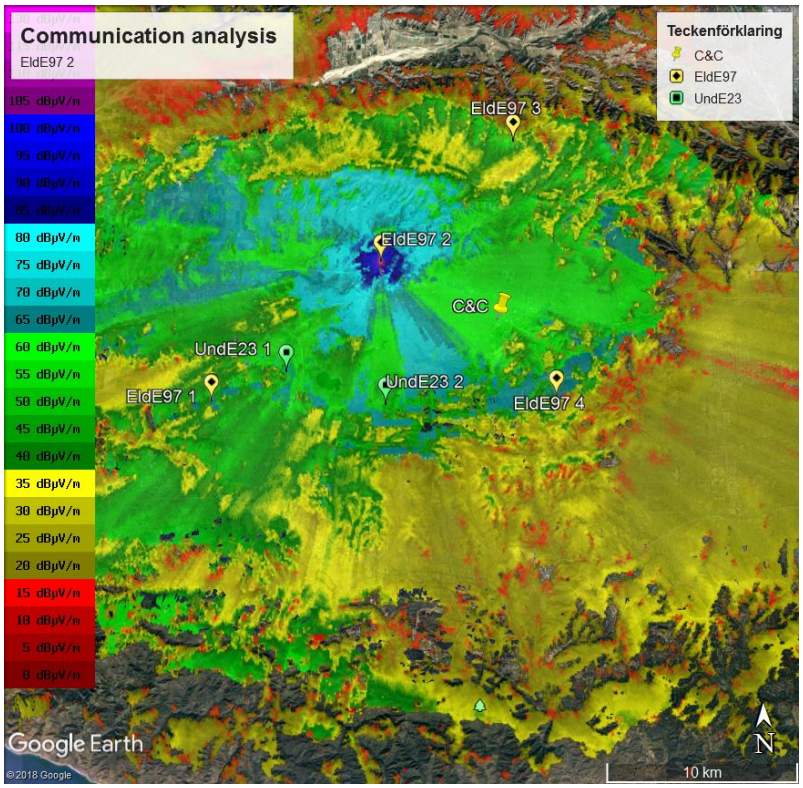


Figure B.10: Radio coverage estimation for the second fire unit.

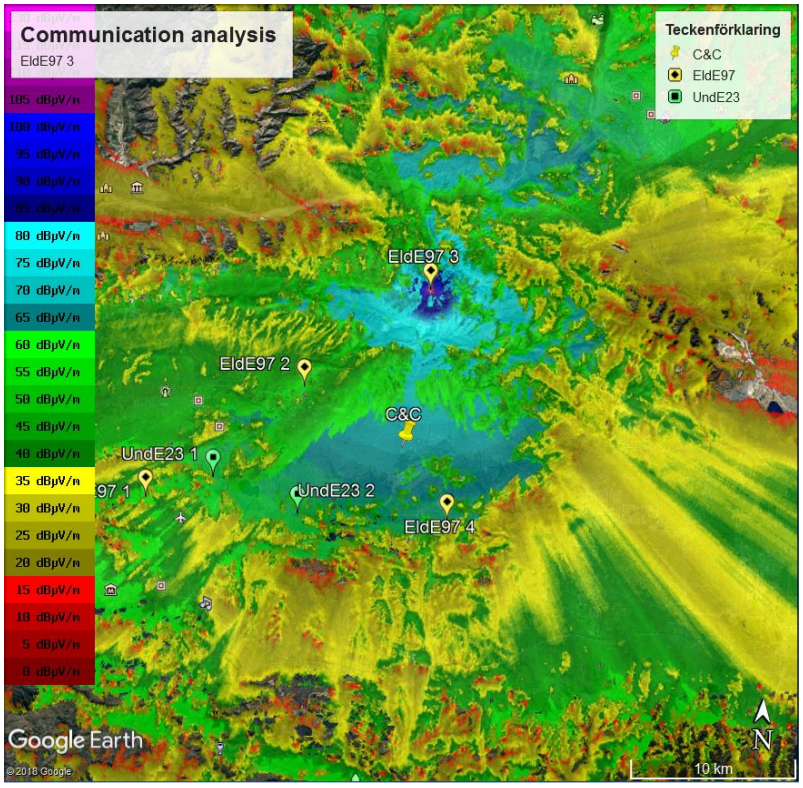


Figure B.11: Radio coverage estimation for the third fire unit.

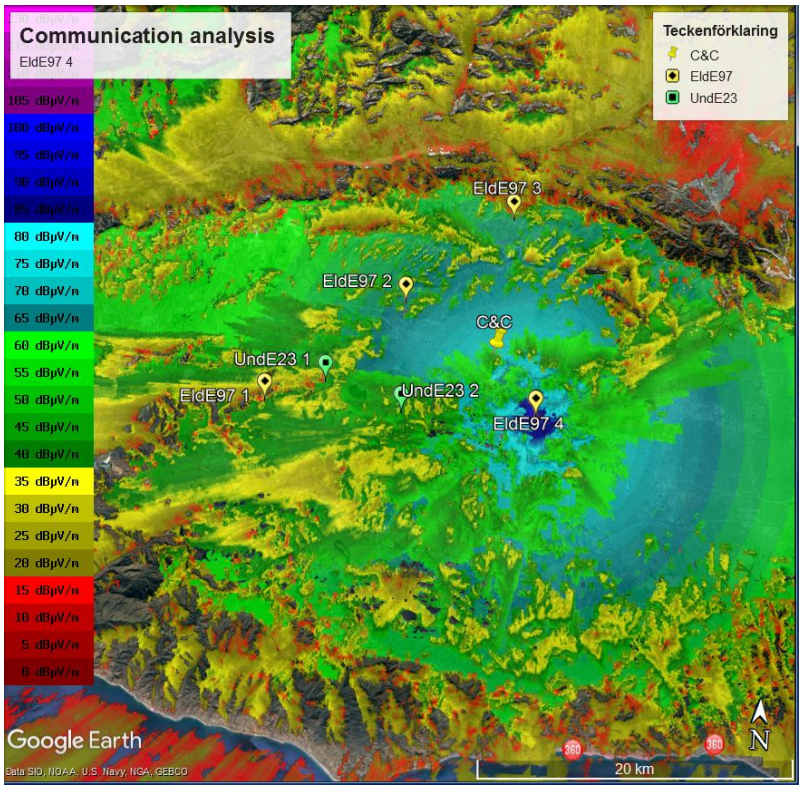


Figure B.12: Radio coverage estimation for the fourth fire unit.