Sérgio Bairos Pimentel

Licenciatura em Engenharia Informática

Drone Route Optimization using Constrained Based Local Search

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Prof. Doutor Pedro Barahona, Professor Catedrático, Universidade Nova de Lisboa

Co-orientador: Prof. Doutor Carlos Damásio, Professor Associado, Universidade Nova de Lisboa

Júri

Presidente: Prof. Doutor José Júlio Alves Alferes

Vogais: Profª. Doutora Paula Alexandra da Costa Amaral

Prof. Doutor Pedro Barahona

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

December, 2018

Drone Route Optimization using Constrained Based Local Search

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Aos meus pais

Acknowledgements

This academic journey has been one of the most rewarding experiences of my life and i am grateful that it occurred on one of the most prestigious colleges in the country. I have found to be true all the good things that were mentioned by so many people about the faculty, whether if it was about the academic environment, the helpful colleagues, the teachers, or the constant vivid activity on the campus. Needless to say, there were some ups and downs, as it normally goes. As such, i would like to thank Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologias for allowing me to grow so much professionally and emotionally as i did the past years. I would also like to thank my adviser, Prof. Pedro Barahona, for guiding me through the elaboration of this thesis and whose help was essential, and my co-adviser Prof. Carlos Damásio for having this interesting thesis subject. To all my class mates that i have encountered throughout the years and that made this adventure much easier and funnier. To my beautiful girlfriend Natacha Teves. Your peacefulness has definitely helped throughout these academic years and i hope i can help you go through yours too. Finally, to my family. To my mother for pushing me to be more disciplined and organized, to my sister for showing me how to be dedicated, to my brother in law for showing me how to be well informed, to my nephew for bringing even more joy to our family, and to my father for showing me how to be ingenious and consistent. Thank you all for giving me so much, and i really hope i can give you so much more in days to come.

Abstract

This dissertation focuses on an optimization problem that consists of finding the best flight plan (i.e. routes) for an Unmanned Aerial Vehicle (UAV), or drone, overflying a farming field that needs to be swept and sprayed with fertilizers or pesticides. This system calculates an optimal route for a crop field, having into consideration that the field needs to be swept and covered entirely. Drones have been around for many years, especially in the military, but only recently they have expanded to other fields, including agriculture, which makes it a particularly interesting field of study. The drones we consider in this thesis are not any regular type of drone for common use, but rather a specially adapted drone for agriculture to facilitate the process of farming. These drones fly close to the ground spraying fertilizers using specialized extension tools. Therefore, the main problem is to search for an efficient route that sweeps the field and saves as much resources as possible. Additionally, the resolution to this problem should not only consist of searching for an ideal route, but also to help the user in planning and selecting a desired field. Some factors need to be taken into account when considering such problems such as legislation, drone's energy consumption, and environment related variables such as field elevations or man-made infrastructures. This system is built using local search as a basis, which essentially consists in making small changes to a solution, improving it iteratively until some near optimal solution is hopefully found.

Keywords: Drone, Local Search, Farming, Optimization, Route, Coverage Path Planning Problem.

Resumo

O tema desta tese foca-se num problema de otimização que consiste em encontrar o melhor plano de voo (i.e. rotas) para um veículo aéreo não tripulado (UAV ou drone em inglês), a sobrevoar um campo de cultivo, que necessita de ser pulverizado com produtos químicos tais como fertilizantes ou pesticidas. Este sistema pretende calcular uma rota ótima para qualquer campo de cultivo, tendo em conta que este campo precisa de ser coberto inteiramente. Os drones já marcaram a sua presença há muitos anos atrás na área militar, mas apenas recentemente é que tiveram uma forte expansão para muitas outras áreas, uma delas sendo a agricultura, o que torna o estudo desta tese particularmente interessante. Estes drones são especialmente adaptados para agricultura e facilitam o processo de cultivo. Sobrevoam os campos a uma altitude relativamente baixa e pulverizam químicos usando extensões mecanizadas incorporadas no próprio drone. O problema principal é, portanto, procurar uma rota eficiente que otimize o número de recursos disponível. Adicionalmente, a resolução para este problema deve não só consistir em procurar a rota ideal, como também fornecer ao utilizador a ajuda necessária para planear e selecionar um campo de cultivo à sua escolha. É necessário ter em conta alguns fatores para este problema tais como legislação, modelo específico do drone em termos de consumo de energia e variáveis ambientais, como por exemplo relevos nos campos ou infraestruturas. O sistema baseia-se na técnica de pesquisa local que consiste em efetuar alterações a uma solução até que eventualmente tenhamos uma solução (quase) ótima.

Palavras-chave: Veículo Aéreo não Tripulado, Pesquisa Local, Agricultura, Otimização, Rotas, Coverage Path Planning Problem.

Contents

List of Figures

List of Tables

Listings

Glossary

CPLEX	An optimization software package developed by IBM.
Hyperspectral	Hyperspectral sensors are devices capable of gathering data using the reflective spectrum of an object. Regular cameras use the visible spectrum for imagery, while hyperspectral cameras gather information from most of the spectrum (visible and not visible), which helps identify important characteristics in the target object.
Instance	In this thesis context, an instance is a data structure that represents the field to be swept and has information about the obstacles, initial/finishing cell and the elevations of each cell.
Load Balance	The load balance algorithm consists in improving the distribution of workloads across multiple resource consumers.
Local Search	The process of iteratively modifying some solution in order to improve it, by doing small changes to that solution.
Metaheuristic	A procedure that helps local search to escape from local optima.

Acronyms

ANAC	Autoridade Nacional da Aviação Civil.
CPP	Coverage Path Planning Problem.
DEM	Digital Elevation Map.
ITS-TSP	Intelligent Transportation Systems Traveling Salesman Problem.
LKH-D	Lin-Kerninghan Heuristic for Drones.
NIR	Near-Infrared.
SA	Simulated Annealing.
TSP	Traveling Salesman Problem.
UAV	Unmanned Aerial Vehicle.
VNS	Variable Neighborhood Search.

Chapter 1

Introduction

1.1 Motivation

In the past, drones were an expensive UAV mostly used by the military. Only recently they have expanded to commercial, scientific, recreational and agricultural applications. In agriculture, the industrial expansion has created even larger crops fields, raising the need to fertilize areas with hundreds of hectares. Adapted manned aircrafts have been used to spray fertilizers or pesticides before, but now the tendency is to make use of drones which makes it more profitable and saves many resources. These types of drones may also be applicable to other situations [7], such as soil and field analysis where at the start of the crop cycle they produce precise 3-D maps for early soil analysis. They may also be used for planting, by shooting pods with seeds and plant nutrients into the soil, providing the plant all the nutrients necessary to sustain life. Another task that drones can help is irrigation, by identifying which parts of a field are dry or need improvements using Hyperspectral, multispectral, or thermal sensors. Health assessment is also important to avoid bacterial or fungal infections, scanning a crop using both visible and Near-Infrared (NIR) light, drone-carried devices can identify which plants reflect different amounts of green light and NIR light. Finally, drones can be used for simple crop monitoring. All these farming tasks face one common problem, the vast fields and low efficiency.

1.2 Objectives

The objective of this thesis is to help the user find the best way to sweep a crop field using a drone with the purpose to spray fertilizers or pesticides. A system should be built that not only provides with a solution, but also aids the user in planning and selecting a desired field to be swept. The system should optimize the coverage route of a drone

in a farming field, taking into consideration four factors which are the terrain elevation, obstacles, the amplitude of the curves made throughout the path and distinct start and finish positions for the route. Additionally, this thesis also addresses a better understanding of optimization problems and the awareness that new and efficient algorithms are needed in today's world, especially in the drone-related field.

1.3 Contributions

Having in mind the set of factors that were just mentioned and the possibility to aid the planning and selection of any field, this dissertation may not only contribute to farming but also to other applications such as surveillance in big facilities where the drone needs to overfly specific areas avoiding some obstacles. Optimization of package delivery routes and recreational usage where the drone must minimize its climbs across steep terrains. Any other application that needs to consider these four factors, are good candidates to benefit from this thesis. Additionally, many researches represent their fields using a grid format with several cells, which the interface of this system is capable of generating, containing information about the real-life field (Instance) such as dimensionality, obstacles, elevations of each cell, and a start and end position for the path.

1.4 Thesis Overview

This thesis starts with a contextualization of the drone route optimization problem in Chapter 2, describing important subjects that should be acknowledged so that it can be easier to understand further chapters. Then, in Chapter 3, an overview of researches elaborated by different authors that helped the making of this dissertation. Chapter 4 describes how the system is composed, detailing its main components and how they are interlinked. Chapter 5 elaborates about the modeling and implementation steps taken to develop one of the system's components, Chapter 6 describes the evaluation and results obtained and finally the conclusions and future work are presented in Chapter 7.

Chapter 2

Context

2.1 SkyScan

Route drone optimization was proposed to SkyScan, a company that focuses in drone engineering. They build their own drones and resell others produced by big companies currently on the market. They provide plenty of flight services such as thermography, cartography, rescues and environment protection, industrial facility inspection and many others. The SS-UR Oracle is a drone designed by SkyScan and can be used in different applications, one of them being agriculture, using a set of extra tools. These tools include a tank with a capacity of thirty litters to fill with chemicals such as fertilizers or pesticides and a pipe extension that is plugged into the drone horizontally and parallel to the ground. This pipe contains small nozzles that allows the chemicals to flow through and consequently spray the crops with the contents of the tank, making it about five-times faster than traditional spraying methods.

2.2 Legislation

Since drone technology has expanded to many other areas, legislation has been trying to keep up with all the sudden conflicts that have been raised. For example, recreational drones have been recently spotted near airport runways, or in other situations where there's high concentration of people like concerts, flying really close to people. In this section we'll briefly focus in Portugal's legislation for generic drone usage established by the Portuguese Civil Aviation Authority, which in portuguese is refered to as ANAC. The published regulation document declares plenty of safety and required rules [2] that need to be considered in certain conditions when operating a drone. All those rules can be summarized into a set of code of conduct norms. The following norms alert the users to

what should not be done relative to a drone's flight plan and operation [1].

1. Do not operate the drone over high concentrations of people (over 12).
2. Do not operate a drone weighting over 25 kilograms without permission of ANAC.
3. Night flights, flights where line of sight is lost, or flights with an altitude of over 120 meters, require authorization by ANAC.
4. Flights over restricted, prohibited, dangerous, reserved or temporarily reserved areas are not allowed.
5. Drones should not fly over certain altitudes defined in areas of operational protection of the national airports without ANAC's permission.
6. Photography and film-making drone operations need to be communicated to ANAC in advance.
7. If the drone is a toy aircraft, it should not operate over people or above 30 meters high.

There's currently no Portuguese legislation specific to drone usage in agriculture, but it is still important do consider the code of conduct just described since it applies to drone usage in general, hence, to farming as well. Also, this thesis will not consider legislation relative to the use of biochemical products such as fertilizers or pesticides.

2.3 The Route Optimization problem

This section describes the problem informally, exemplifies it with a simple scenario and overviews the approach that will be taken to solve such problem. As mentioned, this thesis focuses in optimizing the planning of a drone's route that sweeps an area entirely. This area will possibly have obstacles along the way, steep terrain, a starting point, and an ending point. Suppose that a farmer owns a crop field and wishes to spray fertilizers all over this crop. The farmer takes the drone to the selected starting point, deploys the drone, and uses it to sweep the area while spraying the chemicals. Again, this sweep will need to cover the entire area in such way that minimizes the cost, considering all the four factors the were previously mentioned (obstacles, steepness, path curvature, distinct starting/finishing positions). Figure 2.1 illustrates a possible scenario for the drone to operate.

As described more formally in Chapter 5, the field is represented by a grid with cells (Figure 2.2), that we will assume to have the same size, and should be covered by the drone (the drone covers a cell entirely when it passes over it).

This problem (to sweep an area) is identified as a Coverage Path Planning Problem (CPP). Figure 2.3 is a different example taken from [9]. Their system was built considering two coordinates. However, our approach will consider that each of the cells will also

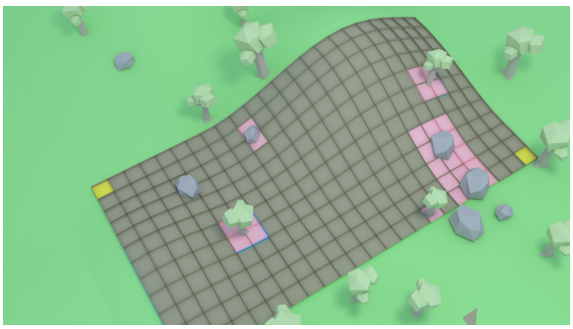


Figure 2.1: An environment scenario. The image illustrates a crop with some obstacles represented as red cells. The yellow cells are the start and end cells in which the drone will initialize and end its flight. The brown cells is the field that the operator wishes to fertilize.

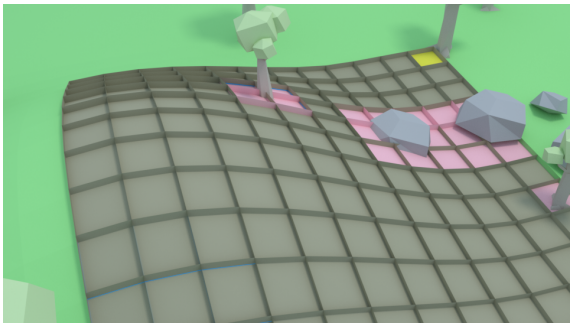


Figure 2.2: Close up of an area. Each area is represented as a grid with cells. It is assumed that the drone covers each cell entirely when it passes through it.

have an altitude component, therefore three coordinates are considered. The approach that will be taken to solve this problem will adopt Local Search using a set of different techniques which will all be described in Chapter 3.

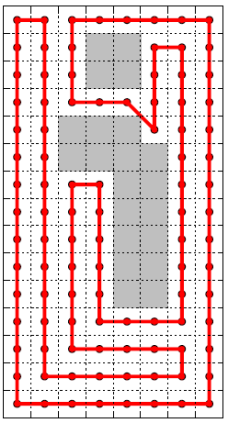


Figure 2.3: Example of a path taken by a drone considering two coordinates and angle of turns.

In summary, the problem to be solved involves the following considerations:

1. The area to be sprayed is, initially, of rectangular shape.
2. There are forbidden cells in the rectangle to represent no-fly zones (i.e obstacles such as tall trees or other man-made infrastructures). These forbidden cells can be used to model the field with shapes different than rectangles.
3. The spraying is done by a single drone which is assumed to cover the whole field in one single route.
4. Throughout the route the drone maintains the same distance to the ground, to maintain the spraying conditions.
5. The elevation of a cell is the elevation of its central point.
6. The drone may pass twice in the same cell. It is assumed that the spraying only occurs in one such passage.
7. The route may start and end in different points, for example in both ends of a road that traverses the field. This is useful since the finishing point serves as a transition point to other fields that need to be subsequently sprayed.
8. The cost to be minimized takes into account the distance traveled by the drone, and is further penalized by its changes in altitude and the curvature of its trajectory.
9. So far, this cost does not consider other potential factors such as the speed of the drone or wind conditions. Neither does it consider the weight loss of the drone during the trajectory due to the spray.

Chapter 3

Related Work

3.1 Coverage Path Planner

The coverage path planning for UAVs has been addressed by recently published works. Modares [9] developed an energy-aware algorithm for area coverage using multiple drones. They modeled the problem by first measuring energy consumption based on distance travelled, speed, and turns taken when performing 45, 90, 135 and 180 degree turns with the drone. Firstly, and more obviously, regarding the distance, the bigger it is the larger the consumption. Secondly, they also measured that, with larger speeds the consumption was actually less because it would take less time to execute the whole path, this may be counter-intuitive but it is what was measured. Finally, increasing the amplitude of the curve increases the energy consumed. With this data they modelled the problem using a cost function that was based in the distance and the turns made by the drone throughout the path. They plugged in this cost function into a Traveling Salesman Problem (TSP) Heuristic, the Lin-Kerninghan Heuristic, which basically transforms an initial non-optimal path into another path that decreases the cost function. These paths are actually tours, which cross all points of interest and finish at the origin point. Then, they distributed the drones across areas using a Load Balance algorithm, in such way that the energy consumption is also minimized to each drone. Their results were compared to other algorithms and succeeded in achieving a better result than those used as benchmarks. Figure 3.1 illustrates their results tested in four different areas, using four different algorithms. Their algorithm is the LKH-D which were compared with a set of other optimization algorithms or software packages such as the CPLEX, DLS and Raster. We'll be using this work as a benchmark, however there are some distinctions. The first being the fact that this dissertation will consider the steepness of the terrain, and the second distinction is that this system will allow to choose different starting/finishing

points for the route.

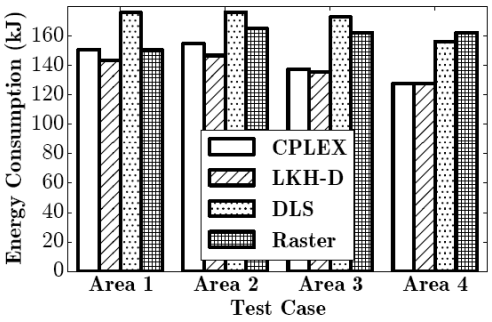
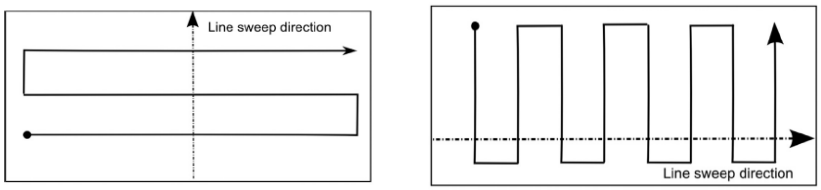


Figure 3.1: Energy consumption on each of the four different type of areas, using four algorithms. Their algorithm is the LKH-D which was successful in comparison to the other three.

Torres [11] proposes a coverage path planning solution for 3D terrain reconstruction. They describe a CPP algorithm that subdivides a complex, concave and polygon-shaped field, into smaller convex subfields. With this, they then find the most preferred trajectory to sweep each of these generated fields. This is refereed to as the line-sweep method and is based on some criteria that allows the UAV to perform less turns in the polygon. For example, considering that a rectangle's largest edge is laying down horizontally (Figure 3.2), a drone would make more turns if it would cover the rectangle back and forth in a vertical fashion rather than horizontal.

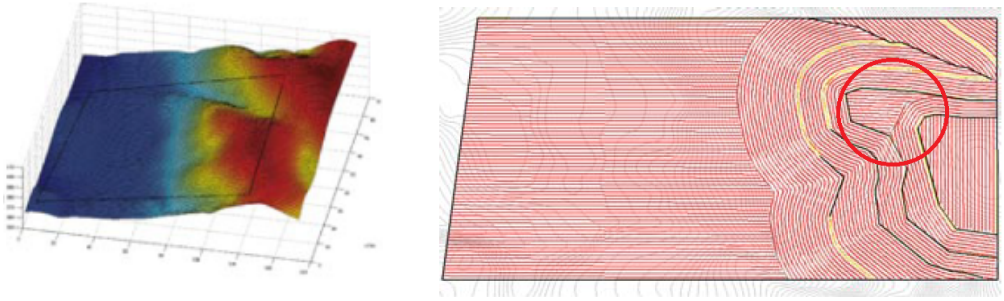


(a) Horizontal sweep (b) Vertical sweep

Figure 3.2: The two types of sweeps in a rectangular field.

Nam [10] developed a different coverage path planning approach for UAVs with the purpose of filming the whole area. They describe a flight planner that generates a coverage trajectory with minimum completion time for quad-rotors. First, they decompose a top-view satellite image into a grid-like representation, where each grid cell size is determined by the camera's field of view. Then they use the wavefront algorithm to determine a navigation function. For the grid generated, they consider the Moore's neighborhood, which means that the drone may take turns of 45, 90, 135 or 180 degrees. In other words, one cell may have 8 neighbors or less, covering the top, sides, bottom and diagonals (instead of Von Neumann neighborhood which only considers top, bottom and sides). Finally, they smooth the previously generated trajectory using the cubic spline method. Di Franco [3] discusses an energy aware coverage path planning solution for single multi-rotor. They derive energy models for different operating conditions based on real

measurements. Jian Jin [5], from an Agriculture Engineering Major point of view, built a system that considers the 3D aspect of a terrain. It starts by modeling the terrain using Digital Elevation Maps (DEMs), then decomposes the terrain in different categories based on multiple criteria such as slope steepness, local elevation variance, slope length, curvature of terrain surface, ground roughness, soil erodibility and so on. Each sub region is then evaluated to define which algorithm should be ran upon the region. Headland turning cost, soil erosion cost and cost from the curvature of the paths are the three most concerned categories of costs used in the problem. However, even though it considers the altitude, the results are not directly applicable for the problem this thesis is addressing. This is because, as can be seen from Figure 3.3, the results obtained consist of multiple sub-regions that were generated, and for each of them a path, which are displayed as red lines. Inside the red circle alone (in sub-figure 3.3b), we can see four different paths which the drone must consider. However, in this dissertation, we will consider that the drone is not autonomous,therefore, a more practical approach will be taken for the operator that is controlling the drone.



(a) DEM (b) Solution

Figure 3.3: The digital elevation map (a) and the corresponding solution (b).

Galceran and Carreras [4] investigated in detail the CPP showing pros and cons of several methods like cellular, grid based, neural network with online or offline computation and for known or unknown areas. Miller [8] proposed an approach to deal with dynamic changes of edge costs in a graph. Their algorithm, the Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP), tries to improve the classical traveling salesman problem (which considers only static costs on runtime). Their analogy to this problem relates to the traffic conditions in a city, where an edge represents the possible path that goes from one location to another, and this cost may vary depending on how much traffic there is at certain time. Except for the work from Jian Jin [5], none of the others (related with CPP) take into consideration the altitude of the terrain. There are not many works being developed regarding the 3D aspect of an area. Additionally, the resolution to the drone route optimization problem developed in this thesis, must consider the ability to include a start point and a finish point for the route, which may be distinct from one another. This helps the operator sweep multiple fields, by defining a finish point in one field in such way that it serves as a transition point for the next field to be swept.

3.2 Local Search Metaheuristics

This section describes local search techniques that may be used to solve the drone optimization problem. Local Search consists in gradually improving a non-optimal solution by doing small changes to it. These changes can also be called moves to the neighborhood (i.e. similar solutions) of some solution currently being evaluated. The problem, however, is that Local Search may find itself in a position where all or most of its neighbors don't lead to any further improvements. Therefore, several techniques have been proposed to avoid this problem. The following sections describe some of those techniques (Metaheuristic). Our problem fits into this set of techniques in a way that allows us to start the system with a non-optimal route (a path), and apply small changes (switch two paths, for example) to improve the previous solution in terms of overall cost. The following metaheuristics are well established in the research community and considered to be some to provide the best results overall.

3.2.1 Tabu Search

The concept behind Tabu Search is to escape local optima by forbidding moves to recently visited neighborhoods. This way, worsening moves can be accepted if no improving move is available which is what may happen in local optima. The Tabu search is implemented with memory data structures that store, for each visited neighbor, a sort of "timestamp" that allows the algorithm to be aware of how old and how recent some of the neighbors have been visited. This way, the recent explored neighbors are unlikely to be chosen in a current search. The main tuning parameter for this metaheuristic is the Tabu table length: the bigger it is, the more extensive "dead-line" for a neighbor is considered, in other words, a recently searched neighbor will only be considered (to be explored again) after a longer period of time.

3.2.2 Variable Neighborhood Search

Variable Neighborhood Search (VNS) consists of diversifying a solution when no improvements have been found. The diversification starts small in the first iterations of no improvements, but if this persists over time, then the diversification is increasingly higher. This scale of diversification is increased until a maximum size, which is denoted as the neighborhood window. Diversifying simply consists in changing or swapping some values in the current solution. This technique allows to escape local optima and explore a wider variety of search options. Regarding the implementation, it basically follows what was just described. The algorithm first proceeds with a greedy local search in some initial solution. If no more improvements have been detected then the diversification method is applied, increasing the scale of it for the next iterations of no improvements. Meanwhile, the best solution is saved throughout the search.

3.2.3 Simulated Annealing

This Simulated Annealing (SA) metaheuristic mimic the real-life process of increasing the temperature of a metal mesh [6]. The metal at high temperatures tends to be much more "malleable" and "receptive" in comparison to low temperature metals that are just harden and tough. The algorithm works the same way as it decides whether it should accept a neighbor solution. When the temperature is high and receptive, a neighbor that worsens the current solution is likely to be accepted, this gives the algorithm the ability to jump out of any local optima it finds itself early on execution. Then, while the temperature is slowly decreased, so is the chance of accepting worse solutions. Hence, the algorithm starts to gradually focus in on an area of the search space. SA takes longer to converge to a good, final solution because, initially, it covers a larger search space instead of an immediate in-depth search. The decision of whether to accept or reject a solution is not only based on the temperature, but also on how worse is the solution compared to the current one. On the other hand, a better solution is accepted unconditionally. In terms of implementation, an initial temperature and cooling rate is set. Then the search is executed until a stop condition is met, which is either when the system has sufficiently cooled down or a good-enough solution has been found. If throughout the search a worse neighbor is considered, then an acceptance probability is calculated based on the temperature and the differences between the current solution and the neighbor solution.

Chapter 4

System Architecture

4.1 Main Components

The system architecture, as shown in Figure 4.1, consists of three main components:

1. **Coverage Path Planner:** This component is in charge of finding the best route to sweep an area. It performs local search on some solution, hence it implements all the three metaheuristics that were previously described.

2. **Planner Debugger/Visualizer:** This is a component that will help understand what the planner is currently processing at runtime by displaying the best route found.

3. **Interface:** This interface will provide real life scenarios/instances to the planner and will be created with Javascript, HTML and Google Maps services.

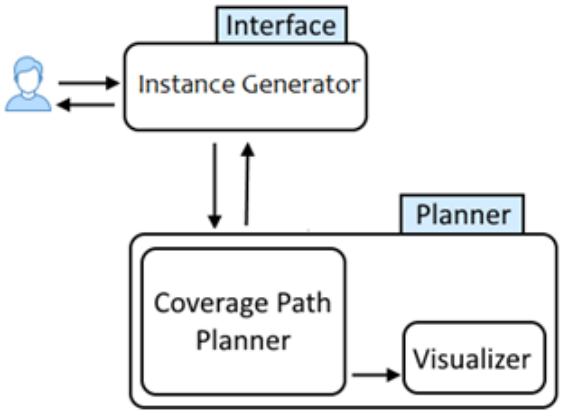


Figure 4.1: System Architecture. The user creates real life instances with the interface which is then provided to the planner.

4.2 Interface

The purpose of the interface is to provide real life scenarios (or instances) to the coverage path planner. Therefore, the application needs to address the following requirements:

1. Inform the user with flight restrictions provided by ANAC.
2. Select a real-life field in Google Maps.
3. Select the obstacles in the selected field.
4. Select the start and ending point for the drone to start and end its flight, respectively.
5. Obtain the elevation of the terrain for the selected field.
6. Generate an instance based on the selected field that can be used in the coverage path planner.

The approach taken to build this application adopts the Google Maps API and services using JavaScript and HTML. The application is illustrated in Figure 4.2.

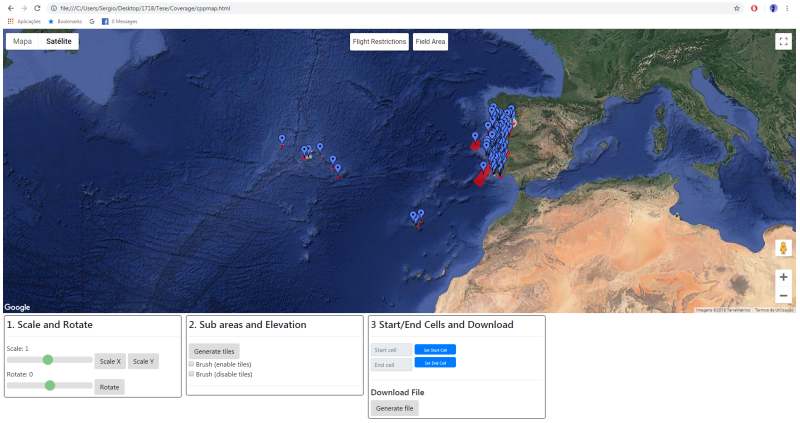


Figure 4.2: The entire application. The map displays the flight restrictions all over Portugal provided by ANAC and a tool bar to aid the field selection and generation. Field Restrictions can be toggled on or off in the top center of the display.

The first requirement is then completed, it informs the user of all the area restrictions and the ability to toggle them on or off. Additionally, if the user clicks in the restriction, it displays the description for that area as shown in Figure 4.3.

The next few steps describe the entire process of generating an instance from a real-life scenario.

Step 1- Area Selection

The first step in the field generation process is to select an area (Figure 4.4). By clicking in any part of the map a pre-defined area is generated. The user must toggle off the flight restrictions if an area inside a restricted zone needs to be selected.

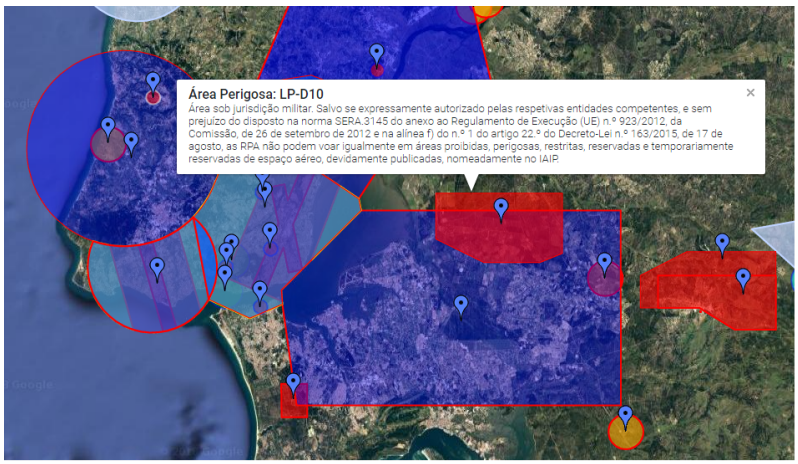


Figure 4.3: Flight restrictions in Portugal. By clicking in a restricted area, the corresponding description is displayed with important information for the drone operator.

(a) Field (b) Area (c) First panel

Figure 4.4: Demonstration of the area (b) being fit onto the desired field (a) using the first panel of the interface (c) to scale and rotate.

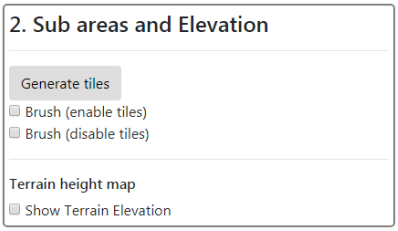
When the pre-defined area is created after the click, the user can now drag, scale and rotate using the first panel named as “Scale and Rotate”. The drag can be made with the mouse.

Step 2 – Tile generation and obstacle selection

Now that the area has been placed, one can now generate the tiles (or cells) as shown in Figure 4.5. It is assumed that the drone covers one cell entirely when passes over it. Given the spray devices of the drones we are considering, the cells have a size of 5x5 meters.

For the tiles to be generated, an area must be created and the “Generate tiles” button must be clicked. Then, the user can click any of the cells to either make it an obstacle or a flyable cell (red cells are obstacles and green cells are flyable cells). To make a faster selection, brushes may also be used (checkboxes in the panel). For example, when the disable brush is active, all it takes to define a set of cells as obstacles is to hover the mouse over the cells. Regarding the elevation of the terrain, a request is sent in the background made by the application and to the Google Elevation Services, obtaining the elevation for

(a) Field



(b) Second panel

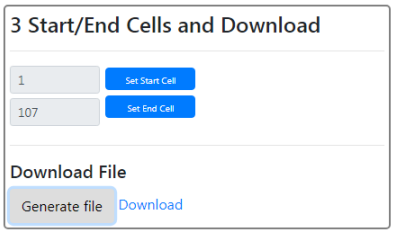
Figure 4.5: Demonstration of the tiles being generated and selection of obstacles (a) using the second panel of the interface (b). Red cells are obstacles and green cells are flyable cells.

each cell that was generated.

Step 3 – Start/End cell selection and file download.

The final step consists of selecting the start/end cell and downloading the file that will be used later in the coverage path planner (Figure 4.6).

(a) Area



(b) Third panel

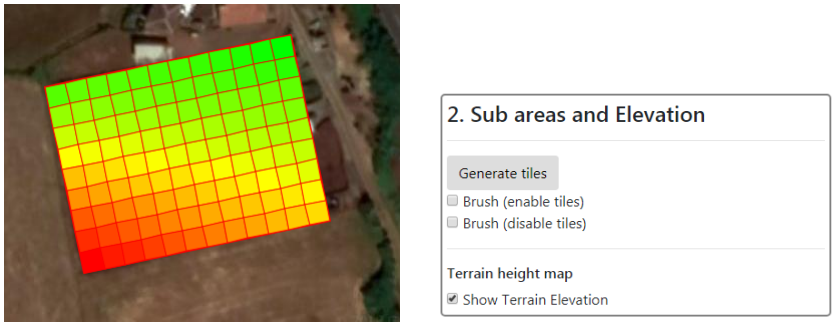
Figure 4.6: Selection of the start and end cell (a) using the third panel (b).

In order to select the start cell, for example, the user must first click in the "Set Start Cell" button, an information message will appear mentioning that the user can now click in one of the tiles to select the start cell. Neither the start or the end cell can be any of the obstacles. Finally, to download the file, all it needs to be done is to click the "Generate File" button and download using the link that will show up right next to it.

4.2.1 Interface pre-processing and other features

Some pre-processing and less relevant features were also included. For instance, assuming the user defines as obstacles an entire row of cells from one of the top or bottom edges in the area, this row will serve no purpose to the final result, and as such, should be removed. The same principle applies to columns in both left and right edges. The removal

of these unnecessary rows/columns makes the result cleaner and helps the performance in the coverage path planner. The user may also use other basic functionalities such as double clicking the area to remove it, or click any other place in the map, creating a new area while removing the previous one. Another useful feature of the interface helps the user choose the start and end positions for the route. The ideal route should preferably begin at the highest point in the terrain and make its way to the bottom, this way the drone is more efficient and does not spend most of its energy climbing.



(a) Area (b) Second panel

Figure 4.7: Elevation of the terrain. Shades of red represent higher elevations while shades of green represent lower elevations.

As seen in Figure 4.7, the elevation is at its highest in the red zone, and the lowest at the green zone. Therefore, the ideal start zone would be somewhere in the bottom left and the finish zone at the top right.

4.3 Planner Debugger/Visualizer

The Coverage path planner debugger/visualizer was developed in order to provide a better understanding of what the planner is processing and was made has a developer/administrator kind of feature, not to be used by the end user. The planner is constantly looking for better solutions, making changes to some current solution to improve it. Therefore, some visualizer would be useful to be constantly reading the contents that the planner is producing and at the same time display it in an understandable way. The visualizer has then the following requirements:

1. Obtain the information from the planner (the best routes found so far).
2. Display the field and the generated routes.
3. Ability to save the best solutions as an image file (displaying the route and the total cost).

The language and framework chosen to build this visualizer was Java Swing. A text file is written from time to time by the planner when a better solution is found. Using an

extra thread, the visualizer checks for recent updates in the text file, and subsequently displays the corresponding route.

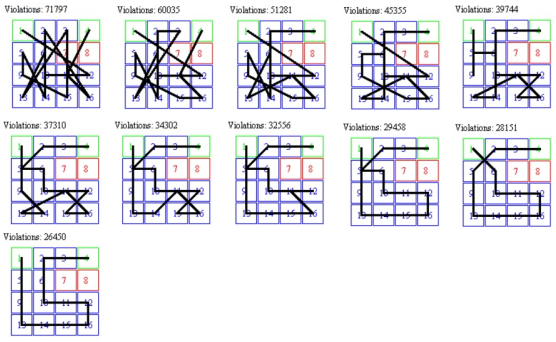


Figure 4.8: The same field instance at different moments in time. The progress occurs from top to bottom, left to right, with decreasing cost at each step. The route is initialized randomly in the first moment of the progress. Cells 7 and 8 are obstacles and are not covered. Cells 1 and 4 is the start and finish position respectively.

Figure 4.8 illustrates the progress made in one simple instance. The visualizer is constantly looking for new routes in the text file populated by the planner and is updated immediately after a new solution is found. In some situations, it can be configured to display only the best solution until some moment in time (instead of all the small changes being made to a solution), since it is beneficial not to overload the file with all the minimal changes that are being made. Additionally, the visualizer has the ability to save the best route and violations as an image file, autonomously. As a side note, although in the first four pictures of Figure 4.8 may appear that the drone flies over an obstacle, the actual path is rather implicit (it takes the shortest path from one cell to another bypassing the obstacles). In other words, the route that is displayed in the image is a shorter version than the actual route. For example, the second sub-figure shows that the route travels from cell 11 to the cell 4, the finishing cell, when in fact it actually travels to cell 4 passing through cell 6 and 2, bypassing the obstacles. This will be formally described in Section 5.3.

4.4 Tools

The coverage path planner was developed using Comet, a programming language designed by Pascal Van Hentenryck and Laurent Michel used to solve complex combinatorial optimization problems in areas such as resource allocation and scheduling. As referred in [13], it offers a range of optimization algorithms: from mathematical programming to constraint programming, local search algorithm and "dynamic stochastic combinatorial optimization". Comet programs specify local search algorithms as two components:

- A high-level model describing the applications in terms of constraints, constraint combinators, and objective functions;

- A search procedure expressed in terms of the model at a high abstraction level.

Its API allows it to be used as a software library. Comet also features high-level abstractions for parallel and distributed computing, based on loop scheduling, interruptions, and work stealing. Regarding the interface of the system, Google Maps services was used with JavaScript and HTML to build a user-friendly interface that lets the user select the desired crops, obstacles and start/end cells to provide to the planner. Java was used to build the planner visualizer. Additionally, other tools may be explored such as OscaR which is a Scala toolkit for solving Operation Research problems and include Constrained Based Local Search techniques.

Chapter 5

Coverage Path Planner

5.1 Description

The coverage path planner is the component that is iteratively searching for new and improved solutions to some provided instance. This planner considers the elevation of the terrain and the amplitude of the curves made throughout the path. The next sub sections describe the approach that was taken to build this planner, including the problem modeling and the necessary implementation steps.

5.2 Problem Modeling

The problem modeling follows a similar approach of [9] described earlier in the related work section, which is a good example to follow and a convenient basis to start from. As mentioned, there are some distinctions between this thesis and the work by Modares. The first being the fact that this dissertation considers the steepness of the terrain, and the second distinction is that this system allows the user to choose different starting/finishing points for the route.

The area to be swept by the drone is represented as a set of grid cells, or nodes, and it is assumed that a grid cell is covered when the drone visits its center. The grid is represented as a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. We let $i,j,k \in \mathcal{V}$ denote a specific node and $e_{ij} \in \mathcal{E}$ denote an edge between nodes i and j. Also, there may exist a set of obstacles $\mathcal{H}$ such that $\mathcal{H} \subset \mathcal{V}$. We'll also assume that the drone always traverses adjacent cells, therefore there will only be edges from one cell to adjacent cells. Figure 5.1 illustrates a simple field represented as a graph.

Denoting (x_i, y_i, z_i) as the Cartesian coordinate of node i, one can now define a cost function which considers the Euclidian Distance between two cells, as in equation 5.1.

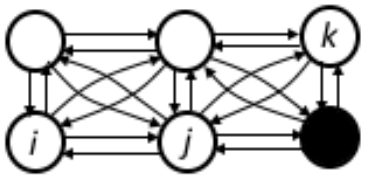


Figure 5.1: Example of an area to be swept represented as a oriented graph. The black node is an obstacle. The graph is oriented since the difference in altitudes are considered.

$$d(i,j) = \begin{cases} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}, & \text{if } e_{ij} \in \mathscr{E} \text{ and } i,j \notin \mathscr{K} \\ \infty, & \text{otherwise} \end{cases} \tag{5.1}$$

With the cost of distances defined, we now need to include the variation of altitude, or elevation costs. To explain the influence of altitude variation, we can imagine having two adjacent cells where the distance between the two cells is the same regardless of the direction it is taking (from A to B or from B to A) but going into higher grounds will have higher impacts in the power consumption, and therefore should cost more. The altitude variation cost function is defined in equation 5.2.

$$h(i,j) = \left(1 + \frac{k_h(z_j - z_i)}{d(i,j)}\right)^2 \tag{5.2}$$

Where z_j is the altitude of the destination cell, and z_i the altitude of the current cell. The term k_h denotes the weight associated with the altitude variation. By having this parameter set to 0, the altitude variation cost has no influence in the overall cost.

As described in [9] we'll also consider the amplitude of the curves made throughout the path. The higher the amplitude made in each turn, more power consumption the drone will consume, which is something we want to minimize as well. Turns vary in 45, 90, 135 and 180 degrees, depending on where the drone is coming from and where it is going to (Figure 5.2). Denoting θ_{ijk} as the exterior angle between nodes $i,j,k \in \mathscr{V}$. The squared length of the edges of the triangle made by nodes i,j,k can be determined as defined in equation 5.3.

$$\begin{aligned} r &= (x_i - x_j)^2 + (y_i - y_j)^2, \\ s &= (x_j - x_k)^2 + (y_j - y_k)^2, \\ t &= (x_k - x_i)^2 + (x_k - x_i)^2 \end{aligned} \tag{5.3}$$

Given the lengths of three sides of the triangle, we can calculate an internal angle using the Law of Cosines. The exterior angle between nodes $i,j,k \in \mathscr{V}$ can be written as shown in equation 5.4.

$$\theta_{ijk} = \pi - \arccos\left[\frac{r+s-t}{\sqrt{4rs}}\right] \text{ radians} \tag{5.4}$$

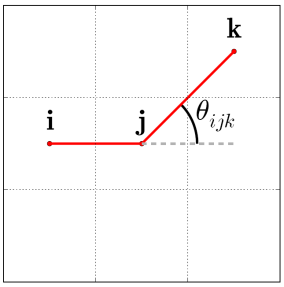


Figure 5.2: A cell and its neighbors, and the exterior angle for three nodes on the path

The term θ_{ijk} is defined in radians, the conversion to degrees is obtained with equation 5.5

$$angle(i,j,k) = \begin{cases} \frac{180}{\pi}\theta_{ijk} & \text{if } e_{ij}, e_{jk} \in \mathcal{E} \\ \infty, & \text{otherwise} \end{cases} \tag{5.5}$$

Now that we know how to calculate the angles given any three nodes, we can define a cost penalization (equation 5.6).

$$a(i,j,k) = \left(1 + \frac{k_a \times angle(i,j,k)}{180}\right)^2 \tag{5.6}$$

The term k_a sets the weight associated with the cost of the angle. With this parameter set to 0, the cost of the angle has no influence in the overall cost. What was mentioned until now in this section, models the problem considering three points:

1. The cost of moving the drone from one point to another, considering the distance and the altitude change, where going to higher grounds will have higher impacts in the cost than going downwards.

2. The cost of performing high degree turns, the higher the amplitude of the curve, costlier will be.

3. The obstacles, where the cost from one cell to some obstacle cell is infinity.

5.3 Implementation

In this section, the equations defined in the previous section related to the three components (distance, altitude and angles) are combined to create a cost system and then further plugged into the coverage path planner. The execution of the planner consists of these seven steps:

1. Reading an instance.

2. Adjacency cost matrix (cost of distances).

3. Floyd-Warshall's algorithm (shortest path between all pairs of nodes).
4. Overall cost matrix (include the penalization of angles and altitudes).
5. Model the planner constraints.
6. Route initialization.
7. Local Search.

Step 1- Reading an instance

The first step the planner must take is to read an instance. A very simple example of the file structure is shown in Listing 5.1 which creates a field shown in Figure 5.3.

Listing 5.1: File structure

```
2 4
1 4
2
2
3
200.24 201.50 202.23 203.24
201.34 202.25 203.50 204.12
```

Each row is listed and described as follows:

1. The number of rows and columns (2 4).
2. The start and end cell (1 4).
3. Number of blocked cells (2).
4. First blocked cell (2).
5. Second blocked cell (3).
6. Elevations of the first row for each column in meters.
7. Elevations of the second row for each column in meters.

1	2	3	4
5	6	7	8

(a) Field

200.2	201.5	202.2	203.2
201.3	202.3	203.5	204.1

(b) Elevations

Figure 5.3: The field (a) and the elevations of each cell (b). The start cell is 1 and the finish cell is 4.

Step 2 - Adjacency cost matrix (cost of distances)

From an implementation point of view, the graph can be represented as an n by m matrix, $\mathcal{M}$, where both rows and columns are uniquely identified by a number from $1, 2 \ldots, n$ and $1, 2 \ldots, m$, respectively. Each entry of the matrix $\mathcal{M}$, denoted by u_{ij} is the cost of going from cell i to j. This second step of the implementation process follows the example shown in Figure 5.4 and consists in populating the adjacency cost matrix (Table 5.1) with the costs of distances. As the name suggests, only the adjacent cells and respective entries are populated with the costs of distances.

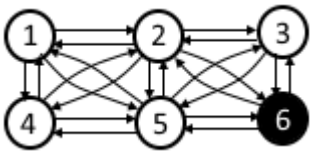


Figure 5.4: A very small example of an area to be swept with one obstacle (cell 6).

	1	2	3	4	5	6
1	0	$d(1,2)$	∞	$d(1,4)$	$d(1,5)$	∞
2	$d(2,1)$	0	$d(2,3)$	$d(2,4)$	$d(2,5)$	∞
3	∞	$d(3,2)$	0	∞	$d(3,5)$	∞
4	$d(4,1)$	$d(4,2)$	∞	0	$d(4,5)$	∞
5	$d(5,1)$	$d(5,2)$	$d(5,3)$	$d(5,4)$	0	∞
6	∞	∞	∞	∞	∞	∞

Table 5.1: The adjacency cost matrix.

Table 5.1 shows the costs of moving from cell i to cell j. Each $d(i,j)$ is defined using equation 5.1. As defined in the equation, we can see from the adjacency cost matrix that the cost is infinity when going from one cell to another that is not adjacent. Additionally, every cell that goes to or from cell 6 (obstacle) is also set to infinity.

Step 3 - Floyd Warshall's algorithm (shortest path between all pairs of nodes)

Now that the adjacency cost matrix has been calculated, the Floyd-Warshall's algorithm [14] is run to obtain the shortest path between any two nodes in the graph. This step also leaves us with an updated matrix (Table 5.2) where the previous entries of non-adjacent cells are now populated with the corresponding costs. For example, the entry $u_{1,3}$ is now defined as $d(1,3)$ since the algorithm gave us the shortest path between the two nodes. In this case, $d(1,3) = d(1,2) + d(2,3)$.

Listing 5.2: Floyd-Warshall's algorithm

```
function S = floyd(M)
  S = M;
  n = size(S,1);
  for i = 1:n  S(i,i) = 0; end
  for k = 1:n
    for i = 1:n
      for j = 1:n
        if S(i,k) + S(k,j) < S(i,j)
           S(i,j = S(i,k) + S(k,j);
  return S;
```

Listing 5.3: Floyd-Warshall's algorithm updated

```
function [S,Next] = floyd(M)
  S = M;
  n = size(S,1);
  for i = 1:n
    for j = 1:n
      Next(i,j) = j
  for k = 1:n
    for i = 1:n
      for j = 1:n
        if S(i,k) + S(k,j) < S(i,j)
           S(i,j = S(i,k) + S(k,j);
           Next(i,j) = Next(i,k);
  return S;
```

To better understand the Floyd-Warshall's algorithm, a brief explanation is given, starting with the basic form of the algorithm shown in Listing 5.2.

The algorithm is described next. Note that the variables defined in this description are refereed only to the variables specified in the code from Listing 5.2.

1. Initialize a matrix S of shortest paths with the adjacency matrix (the matrix calculated in the previous step). Nodes that are not adjacent have infinity cost in the corresponding matrix entry.

2. For all values k from 1 to n iterate. On iteration k, update S, by considering all indirect paths passing through node k.

3. After the last iteration the costs of distances between any two nodes in the graph are stored in matrix S and returned.

This algorithm only computes the costs (distances) of the shortest paths between any two nodes but does not return the actual paths. To find these paths we first need to start by adding a little bit of code. Listing 5.3 shows us the updated code.

Listing 5.4: Path reconstruction

```
function P = path(u,v,Next)
  if N(u,v) == inf
    P = [];
    return;
  end
  P = [u];
  while u != v
    u = Next(u,v);
    P = [P,u];
  end
end
```

Now, we have included the matrix $Next$ (as in next node). This matrix is initialized with j, i.e. it assumes that a direct path from i to j with no intermediate nodes is the best choice (so far). Then, in the inner loop of the Floyd-Warshall's algorithm, if a new shortest path is found, the new leading arc is updated accordingly. The final $Next$ matrix basically tells us that, if we wish to go from cell A to B, then we must start the path by first going into cell C. Consequently, there is still a piece of code missing, which is the code that reconstructs the entire path of going from one cell to another. This code is shown in Listing 5.4

The code in Listing 5.4 shows us that the path between any two nodes u and v, can be obtained by following the trail indicated in matrix $Next$. The first test checks whether there is an actual path between nodes u and v. If there is a path, then the reconstruction begins, starting in node u.

	1	2	3	4	5	6
1	0	$d(1,2)$	$d(1,3)$	$d(1,4)$	$d(1,5)$	∞
2	$d(2,1)$	0	$d(2,3)$	$d(2,4)$	$d(2,5)$	∞
3	$d(3,1)$	$d(3,2)$	0	$d(3,4)$	$d(3,5)$	∞
4	$d(4,1)$	$d(4,2)$	$d(4,3)$	0	$d(4,5)$	∞
5	$d(5,1)$	$d(5,2)$	$d(5,3)$	$d(5,4)$	0	∞
6	∞	∞	∞	∞	∞	∞

Table 5.2: The updated cost matrix.

Step 4 - Overall cost matrix (include the penalization of angles and altitudes)

In step 2, the cost matrix $\mathcal{M}$ was populated with the adjacency distance costs. Then in step 3, this matrix was used to execute the Floyd Warshall's algorithm, which gave us the shortest path between any two nodes in the graph. Having these shortest paths, we can now update our matrix to include both the altitude variation costs and the cost relative

to the amplitude of the curves made throughout the path. Therefore, this fourth step consists in updating the previous matrix with the costs just mentioned.

Given the shortest path between any two cells of the grid (nodes a and b, for example), let's denote the path between these two nodes as a sequence of r cells, P_{ab}, where each cell of P_{ab} is defined as p_i and $1 \leq i \leq r$. p_1 is the start node (a) and p_r the finish node (b) of the path sequence P_{ab}. With this notation, the terms $d(a,b)$ from Table 5.2 are updated to $t(a,b)$ using the formula shown in equation 5.7.

$$t(a,b) = \sum_{i=1}^{r-1} d(p_i, p_{i+1}) \times h(p_i, p_{i+1}) \times a(p_{i-1}, p_i, p_{i+1}), \text{ such that } p_i \in P_{ab} \tag{5.7}$$

The total cost matrix is then completed and can now be used by the planner. To exemplify, let's try to obtain the cost of going from cell 3 to cell 4. The shortest path sequence from node 3 to 4 is: $P_{3,4} = (3,5,4)$. Equation 5.7 breaks down into the following segments: $t(3,4) = d(3,5) \times h(3,5) \times a(0,3,5) + d(5,4) \times h(5,4) \times a(3,5,4)$. We assume that p_0 is a virtual previous point so that every $a(p_0, x, y)$ equals 1.

Step 5 - Building the Model

Comet allows us to build a constraint system that helps the planner obtain better solutions, measured by the degree of violations of the specified constraints. By imposing a constraint that the cost should be zero, the solver indirectly minimizes the cost by minimizing the constraint violation. Having in mind that a solution is a sequence of steps between cells, the overall constraint violation is the sum of the violations of the individual steps (each constrained to have zero cost). When the steps involve non-contiguous cells, care must be taken regarding the penalization of the curves. For example suppose we ended up with the solution (1,3,5,4,2) for the field of Figure 5.4. In this case the angle to be considered when going from cell 3 to 5 must take into account that the drone comes from cell 2 and not cell 1.

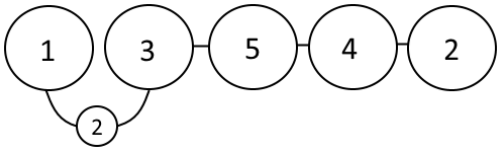


Figure 5.5: A non optimal solution. The path taken from cell 1 to 3 is through cell 2. From 3 to 5 is direct since they are adjacent.

Step 6 - Route initialization

Every local search procedure must start with some solution. Therefore, two distinct heuristics were used and compared for the initial route:

1. **Randomized:** The initial route is completely randomized.

2. **Field Sweep:** The initial route sweeps the area horizontally or vertically along the longest dimension.

Step 7 - Local Search

With everything set up, the local search is initialized. For this step, all three meta-heuristics described earlier in Section 3.2 were used and tested, but for Tabu and VNS a very small change was made regarding its implementation, as described next. Local search selects, at each iteration, two cells to swap and change routes. This is the well-known 2-opt algorithm which was adopted in this dissertation. As described in [12] the main idea behind 2-opt is to take a route that crosses over itself and reorder it so that it no longer does. For example, if the current solution for some basic field is (1,4,6,7,9) and cells 4 and 9 are selected, then the changed and improved solution will be (1,9,7,6,4). All the cells between 4 and 9 are also mirrored to the opposite side. The cell selection step may be performed in various ways. The ideal way is to select two cells that minimize the overall cost, but this can be very costly in terms of computational time because the minimum of all possible combinations must be found. A faster approach consists of first randomly selecting one cell and then select the second cell that minimizes the cost. This helps to detect a wider variety of solutions faster but may be harder to find the optimal solution, and at some iterations an improvement may not even be found because the selection of the first cell was not adequate. The adapted VNS is basically the same as the original but with changes regarding the cell selection. It consists of performing the first type of cell selection (two cells that minimize the cost), and if the time that took to select those two cells passes some pre-defined threshold, then the selection type will change for the next iterations to the second type (first select a random cell, then select a second one that minimizes). As it was mentioned, using this second type of selection may lead to iterations where no improvement was made at all, not because it reached a local optimum, but because the first randomly selected cell did not lead to any further improvements, while there may be other cells that would. For this reason, a new variable was added only for the second type of cell selection. This variable is called the fail streak variable. If no improvements were made in the current iteration the fail streak is increased until it reaches the maximum streak. After it has reached the maximum streak, the normal VNS diversity method is applied. Regarding Tabu Search, it follows the same cell selection principle as the adapted VNS, the rest of the algorithm is the same as described in Section 3.2. SA doesn't need to find two cells that minimize the costs, it evaluates some neighbor solution and decides whether it should accept it or not, hence, this fail streak variable is not necessary.

5.4 Scaling the planner

The problem of executing local search in much bigger instances is the search space itself and the time it takes to find the solution. It's just not viable to feed the solver with such big instances. Therefore, a technique was prepared that breaks down the main problem (a very big instance/field) into smaller problems (smaller instances/fields). This technique consists of creating sub-areas from the initial field as shown in Figure 5.6

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104
105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120

(a) Field

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88
89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104
105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120

(b) Sub-areas

Figure 5.6: The field instance from the paper chosen as a benchmark [9] with obstacles as grey cells. On the left the normal field and at the right all the 8 sub-areas generated delimited as different shades of green.

The algorithm starts by selecting an unexplored node (let's say cell 1) and expands diagonally at each step. When the expansion reaches cell 19, the area has the dimensionality of 3x3 units. From there, it detects that it can no longer expand diagonally because of the obstacle in cell 20 and, as such, it should continue expanding vertically, to cell 27, concluding a new area of 4x3 units. This process is repeated until all cells are explored. The purpose of this is to create a planner for every generated sub-area. The solutions of each area would need to be merged in such way that the end point of one sub-area matches the start point of the next. Only the sub-area generation was developed, hence, the technique was not integrated with the current planner, and is left out as future work.

Chapter 6

Results

This chapter analyses the results obtained from a series of tests using all the components of the system's architecture. A comparison analysis is also described based on the selected benchmark [9] (Figure 2.3). It is important to know the impact of the components (elevation and angle) included in the cost system of the planner and what type of system configuration is preferred in order to obtain the best possible solutions.

6.1 Tests Structure

Before describing each test individually, we will first mention the main parameters that are included in this system and that are meant to be tested. The different parameters are the following:

Parameters

1. **Instance(s):** The set of instances/fields that we wish to cover.
2. **Metaheuristic (h):** The metaheuristic used in local search as described in Section 3.2. (SA, VNS and Tabu search).
3. **Metaheuristic parameter value (hval):** For each metaheuristic, the main tuning parameter will be evaluated with different values.

   SA: Initial temperature, with possible values of 10, 25 and 50.

   VNS: Maximum neighborhood window. Which varies from 25%, 50% or 75% of the total number of flyable cells in the instance.

   Tabu: The Tabu table length (Tenure), with values of 10, 15 and 20.

4. **Initial route (ir):** The initial route (Section 5.3), which it's either Field Sweep or Randomized route.

5. **Elevation weight (kh):** The weight given to the elevation of an instance (Equation 5.2). The possible values are 0%, 10%, 33%, 75% and 100%.

6. **Angle weight (ka):** The weight given to the turns made in some route of an instance (Equation 5.6). The possible values are the same as for the elevation weight: 0%, 10%, 33%, 75% and 100%.

7. **Number of tries (o):** The number of tries for each setting established with the previous parameters.

8. **Total time (l):** The total amount of time to run for each setting established with the previous parameters, in milliseconds .

Using the previous parameters, one can now configure multiple types of tests. The following tests were chosen:

Tests

1. **Metaheuristic test with no weights:** The purpose of this test is to analyze which metaheuristic (Section 3.2) performs best. The parameters used for this test is the initial route and the metaheuristic parameter value. The weights of both elevation and angle costs are not included.

2. **Weighted Metaheuristic test:** For this test, the same parameters of the previous test apply. Additionally, the weights of the altitude and angle cost are considered.

3. **Benchmark test:** The purpose of this test is to configure the system in such way that it resembles the system of the benchmark. This means that only the angle cost component is considered with no elevations in the instance. The best metaheuristic obtained from previous test was used. Additionally the initial route is also tested.

4. **Case scenarios:** Using the interface, different types of real life scenarios are used and tested in the planner using the best configurations obtained from the previous tests.

6.2 Metaheuristic test with no weights

As mentioned, this test has the purpose to discover which of the three meta heuristics are adequate for the route optimization problem. In this case, the feature that we're looking for in a meta heuristic is which of the three can get best results in a reduced amount of time. Some heuristics are reasonably good when used for short time periods but may not

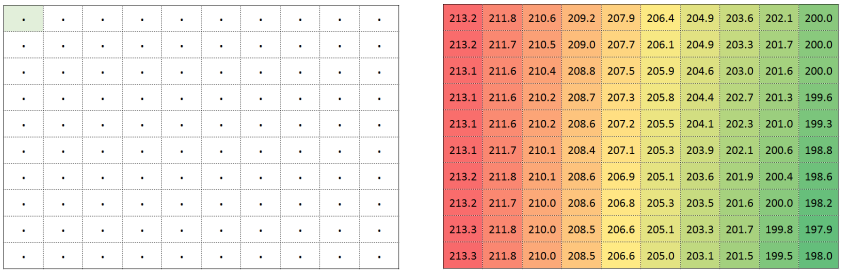
get the best of solutions, while others are meant to search for a wider search space, which increases the chances of finding the optimal solution, but will take higher amounts of time. The configuration used for this test was the following:

1. **Instance:** field_10x10_basic.txt (Figure 6.1)
2. **Metaheuristic (h):** All three metaheuristics described described in Section 3.2 (VNS,SA,Tabu).
3. **Metaheuristic parameter value (hval):**

   SA: 10, 25, 50

   VNS: 25%, 50%, 75%

   Tabu: 10, 15, 20
4. **Initial route (ir):** sweep, rand
5. **Elevation weight (kh):** 0
6. **Angle weight (ka):** 0
7. **Number of tries (o):** 3
8. **Total time (l):** 300000 ms (5 minutes)



(a) Basic field (b) Elevation

Figure 6.1: The basic field. This field is 10x10 units long and has no obstacles. The elevation was obtained with the interface by dragging the 10x10 area onto a random location. The start and ending point is the top left cell.

As can be seen from the graph of Figure 6.2, VNS clearly wins in terms of average cost, while SA and Tabu stand close to each other but with Tabu achieving better results. The difference between SA and VNS is that VNS will immediately start looking for the best possible moves, leading to a local optimum, where it will try to diversify and reach to other solutions from that position. On the other hand, SA prefers to slowly get to know the entire neighborhood before diverging into a local optimum, and as such, may take longer to do so. Tabu can be seen as a mixture of both, it tries to reach to a local optima fast but may accept worsen moves along the way. Overall, the field sweep method for the

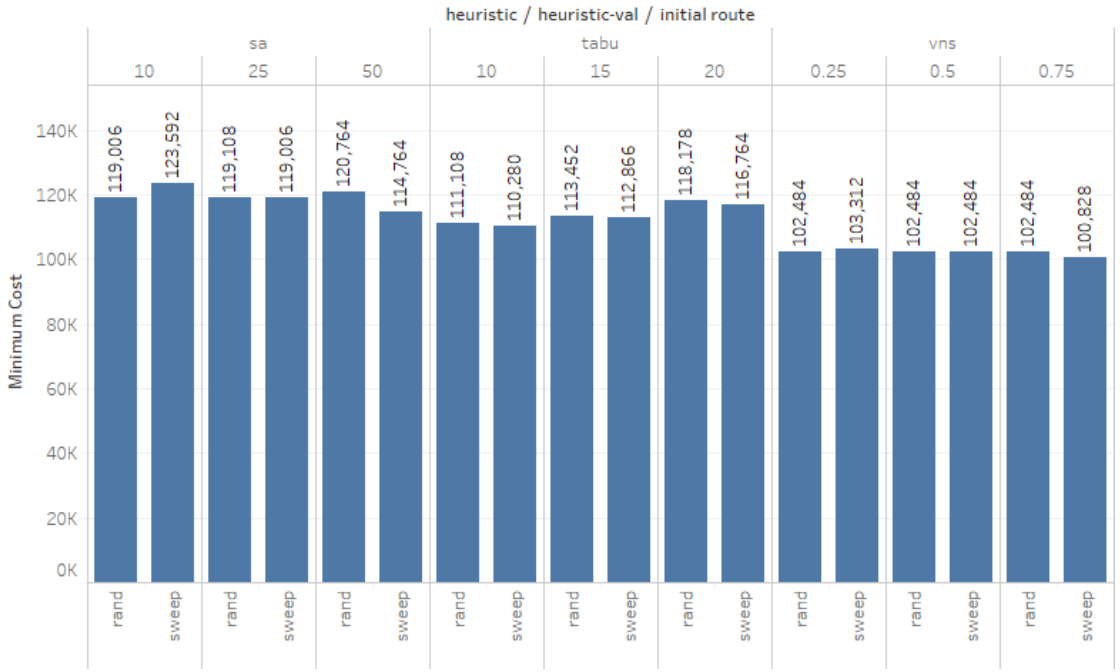


Figure 6.2: The results for the metaheuristic test. The vertical axis represents the minimum cost of all tries with the respective configuration and the the horizontal axis (top) the meta heuristic used with corresponding parameter values. The bottom horizontal axis is the initial route.

initial route is the method that obtains the better results. Intuitively, one may think that field sweep is the best method for every case, since it resembles with what it is trying to be achieved in crop sweeps, but that may not be always the case since this method will most likely lead to a search space that is biased by the initial route. Therefore, sometimes the randomization method may be better since it allows to cover a wider variety of search areas, even though most of these areas don't lead to the best solutions.

6.3 Weighted Metaheuristic test

As described, the weighted metaheuristic test is the same as the previous test but with the addition of testing the weights of both components, the weight of the angle cost component (ka) and the weight of the altitude cost component (kh). The test is then configured as follows:

1. **Instance:** field_10x10_basic.txt (Figure 6.1)
2. **Metaheuristic (h):** All three metaheuristics described described in Section 3.2 (VNS,SA,Tabu).
3. **Metaheuristic parameter value (hval):**

   SA: 10, 25, 50

VNS: 25%, 50%, 75%

Tabu: 10, 15, 20

4. **Initial route (ir):** sweep, rand
5. **Elevation weight (kh):** 0, 0.1, 0.33, 0.75, 1
6. **Angle weight (ka):** 0, 0.1, 0.33, 0.75, 1
7. **Number of tries (o):** 3
8. **Total time (l):** 300000 ms (5 minutes)

We can see from Table 6.1 that VNS still performs better under different weight configurations. To better understand the comparison between each metaheuristic, Table 6.2 displays some auxiliary results. This table basically displays how effective a metaheuristic has been when compared to another. For example, in the first row, we can see that Tabu search achieved better results on 15 out of the 25 possible combinations of weights when compared to SA, and that the average gains in cost of using tabu is 25414. With this, the results let us decide that VNS is the preferred metaheuristic for this problem and the only one used in further tests.

Table 6.1: The weighted metaheuristic test with different altitude and angle weights. For each combination, the minimum cost is displayed from a series of tries.

Min. viols: 100,828 – 528,626

		kh				
heuristic	ka	0	0.1	0.33	0.75	1
sa	0	114,764	112,604	124,071	154,313	174,788
	0.1	121,894	117,469	133,758	165,828	197,017
	0.33	125,569	133,827	157,727	213,466	257,556
	0.75	148,188	168,582	248,977	363,662	411,680
	1	192,300	202,488	266,715	422,099	528,626
tabu	0	110,280	111,925	124,665	146,544	162,952
	0.1	120,875	121,966	135,411	166,428	181,039
	0.33	136,247	137,368	160,565	201,391	214,298
	0.75	172,625	178,752	207,983	279,819	276,974
	1	196,888	197,044	233,814	322,707	324,056
vns	0	100,828	101,673	107,577	123,410	138,827
	0.1	102,369	104,166	110,185	127,693	144,219
	0.33	106,783	107,062	109,646	137,783	161,583
	0.75	116,929	116,599	141,875	200,590	262,774
	1	123,750	124,999	153,712	286,914	329,372

Table 6.2: Heuristic comparison.

Heuristics	Avg viols	Count
tabu:sa	25 414	15
vns:sa	64 666	25
vns:tabu	39 252	24

6.4 Benchmark test

The purpose of this test is to compare our solution with the one from the benchmark (Figure 2.3) obtained in [9]. Again, there are some distinctions between the two systems. The first being the fact that the system of this dissertation considers the steepness of the terrain, and the second being that it also allows to choose different starting/finishing points for the route. The cost functions used are also different. Their approach involved in separating both components costs (angles and distances) and then summed those two for the final cost. This was possible since they were based on a drone's consumption model, in other words, they know how much power consumption is consumed when making turns alone (while standing still in the same position) and the power consumption while traveling through distances in straight line. On the other hand, this system uses raw cost functions not based on a model, hence, the cost system uses the distances as a basis and then a penalization multiplier is applied to it based on the elevation costs and angle costs. This aproach gives us a generalist idea of how the system behaves on any model, and some model could be included in the future, as it will be mentioned in the future work section. With this said, for this test, a configuration of parameters were used that resembles their system, which means that no altitude is considered and the start and finishing position must be the same. Only the best metaheuristic (VNS) from the previous tests are used for this test. Since the number of parameters are now less, the number of tries have been increased from 3 to 5. The initial route is also included. In summary:

1. **Instance:** bench.txt (Figure 2.3)
2. **Metaheuristic (h):** VNS
3. **Metaheuristic parameter value (hval):**

   VNS: 25%, 50%, 75%
4. **Initial route (ir):** sweep, rand
5. **Elevation weight (kh):** 0
6. **Angle weight (ka):** 0, 0.1, 0.33, 0.75, 1
7. **Number of tries (o):** 5
8. **Total time (l):** 300000 ms (5 minutes)

From Figure 6.3 we can observe that the higher the weight of angle component (ka), the greater the incidence of the angle cost in the overall cost. The distances cost tend to be the same even with different values of ka, this occurs because of two reasons: The first being the fact that the solver will always try to pass through one cell only once, since passing through one cell more than once means that the path is not properly optimized, unless it really must take that path. Hence, the total distance of an optimized path will be

closely related to the estimated distance based on the number of cells the field contains. If, on top of this, we add the angle costs, we are basically telling the solver to minimize those turns, but in the end, it still has to cover the same cells, and the solver will still try to pass through one cell only once. Therefore, the small difference of distance costs between each *ka* in the graph is due to the fact that the diagonals that occur in the path cost slightly more, and as such, some solutions may contain more curves in the diagonal.

Additionally, this other graph in Figure 6.4 shows us that the average number of turns (in degrees) in a solution is decreased with higher values of *ka*. However, when *ka* is 1 the average number of turns is slightly higher. This is because we have reached an approximate minimum average of turns for the solution. The slight differences between the weights 0.33. 0.75 and 1, are caused by the diversification step of VNS which is associated with the metaheuristic parameter value (25%, 50% or 75%) and the randomization of the initial route.

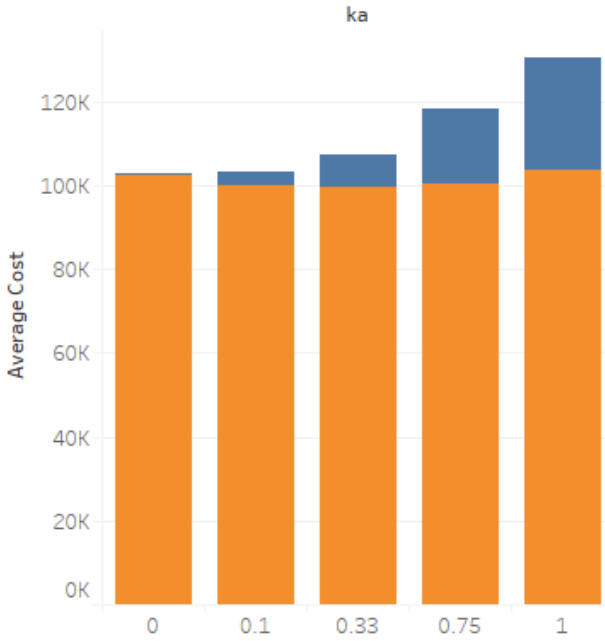


Figure 6.3: The impact of the angle (blue) and distance (orange) costs in the overall cost for different values of *ka*.

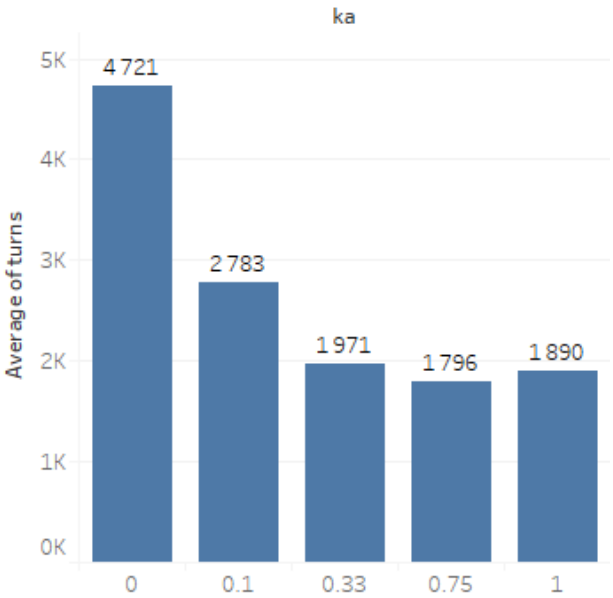
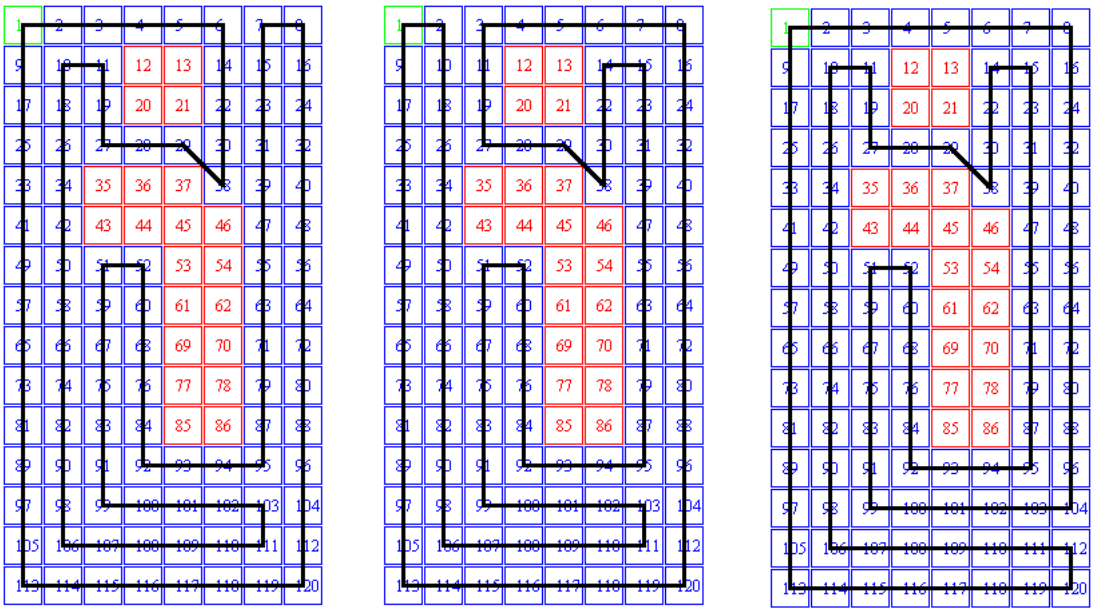


Figure 6.4: The average number of turns (in degrees) of a solution for different values of *ka*.

The best results are displayed for this test (Figure 6.5) and the corresponding configuration of parameters that were set when the solution was obtained (Table 6.3). The solution in sub-figure 6.5b is the solution obtained in the work chosen as benchmark [9]. All three solutions have the same number of turns (tt). Also, solution a) has lower cost, but that is only because a smaller *ka* value was used (0.33) and therefore the angle cost impact is smaller in the overall cost. All three solutions are viable solutions and should be acknowledged by the end user to decide which one to take. The table also informs us that the randomization of the initial route is a good heuristic since it was used in two of the solutions. This may be due to the fact that the field has multiple obstacles. Table 6.4 shows us that the randomization heuristic was actually slightly better than field sweep. In further tests, we will see if these results still persist when using a field with no obstacles.



(a) (b) (c)

Figure 6.5: The best solutions for the benchmark test.

sol	cost	h (hval)	ir	*ka*	dc	ac	tt
a	105925	vns (0.5)	rand	0.33	99414	6511	1620
b	115691	vns (0.5)	sweep	0.75	99414	16277	1620
c	115691	vns (0.75)	rand	0.75	99414	16277	1620

Table 6.3: The configuration of parameters that were set when the different solutions were obtained. The last three columns stand for distance cost, angle cost, and the sum of turns taken (in degrees), respectively.

ir	average cost
rand	112125
sweep	112867

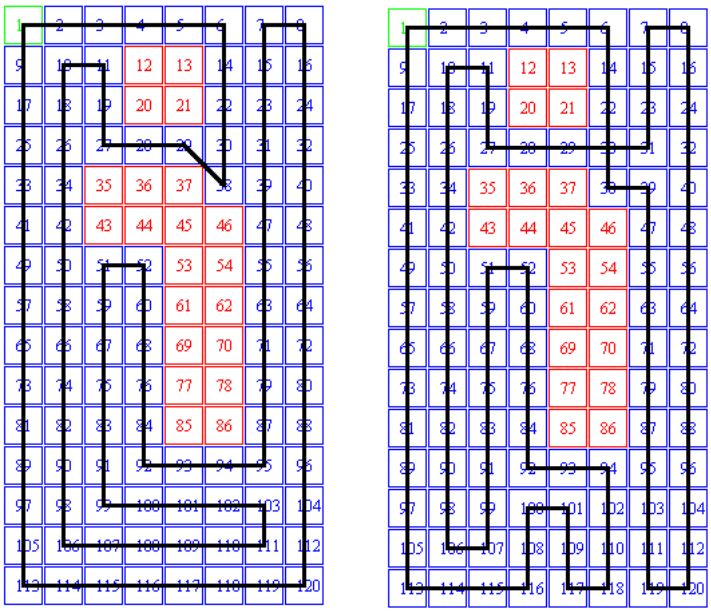
Table 6.4: Initial route comparison for the benchmark test.

6.4.1 Consistency test

This small test is intended to analyze the consistency of results when the same configuration is used. Only the configuration of solution 6.5b is set (Table 6.3) and a total of 10 tries were executed.

try	cost	variation	h (hval)	ir	*ka*	dc	ac	tt
1	120493	4.2%	vns (0.5)	sweep	0.75	100000	20493	2070
2	115691	0%	vns (0.5)	sweep	0.75	99414	16277	1620
3	118499	2.4%	vns (0.5)	sweep	0.75	100243	18257	1800
4	119390	3.2%	vns (0.5)	sweep	0.75	100243	19148	1890
5	118499	2.4%	vns (0.5)	sweep	0.75	100243	18257	1800
6	115691	0%	vns (0.5)	sweep	0.75	99414	16277	1620
7	117473	1.5%	vns (0.5)	sweep	0.75	99414	18059	1800
8	116929	1.1%	vns (0.5)	sweep	0.75	100000	16929	1710
9	120281	4%	vns (0.5)	sweep	0.75	100243	20039	1980
10	117473	2%	vns (0.5)	sweep	0.75	99414	18059	1800

Table 6.5: The results obtained from all the tries (10) where each try has a maximum runtime of 5 minutes. The last three columns stand for distance cost, angle cost, and the sum of turns taken (in degrees), respectively. The variation column tells us how different the solutions are when compared to the best in terms of overall cost.



(a) Best (Try 2) (b) Worst (Try 1)

Figure 6.6: The best and worst solutions obtained from the consistency test.

From both the consistency results table (Table 6.5) and the path taken in the best and worst tries (Figure 6.6) we can observe that the consistency of results is quite good overall since the worst is quite similar to the best solution. The easiest way to see which solutions are the best is to check whether the solution has made many turns in the path, this is done by looking at the tt column. This column shows us the sum, in degrees, of all the turns made in the path, therefore, the less turns a path makes, the most likely that the solution is one of the bests.

6.5 Case scenarios

The following sections describe different cases obtained from real life scenarios using the interface. The configuration of some parameters for all the scenarios is now fixed. The parameters are the best metaheuristic (VNS), and its corresponding value (0.75) for the main parameter.

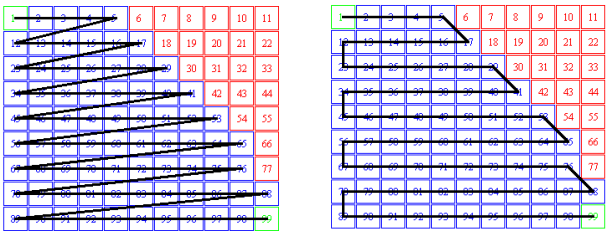
6.5.1 Simple scenario

This simple scenario is an example of how the planner obtains good solutions for types of fields that are most common in a real life scenario, which are fields with no obstacles in the middle of it and where the elevation is pretty much flat. For this reason, this case will not give much emphasis to the elevation costs. As shown in Figure 6.7, the interface is used to generate the desired field.

(a) Field (b) Area (c) Tiles

Figure 6.7: The simple scenario. The field is 9x12 units long. In (c), the start point is the top left yellow point and the finish is the bottom right point.

The field has only one obstacle in the top right corner, which is used to delimit the area. As mentioned in Section 4.2.1, the interface also takes care of clearing out edges where only obstacles can be found. In this case, the first column from the right edge is only defined as obstacles and therefore doesn't have any use to the final result. This column is then removed, and the final instance is converted from 12 columns to 11 (Figure 6.8).



(a) Initial route (b) Solution

Figure 6.8: The initial route (a) using field sweep, and the solution (b).

From Figure 6.8 we can see that once the initial route is created using the field sweep method, the solver will take care of the remaining changes to improve the solution. Again, it may appear that the route in Figure 6.8a goes directly from cell 6 to 12 in that specific

manner shown in the image. But in fact the actual route is implicit and the drone will always take the shortest possible route traveling across complete cells. For example, the shortest route between cell 6 and 12 could be (6,4,14,13,12). In other words, the route displayed in the image is a shorter version than the actual route. Previously in the benchmark test, the analysis of which initial route was better led us to a conclusion that the randomization method was slightly better than field sweep. We also study whether the results persist for this type of field (no obstacles in the middle) in Table 6.6.

ir	average cost
rand	101311
sweep	88959

Table 6.6: Initial route comparison for the simple scenario.

This time, the initialization of the route using the field sweep method achieves better solutions than the randomization method. This allows us to conclude that the randomization method is more adequate for fields with multiple obstacles located somewhere in the middle of the field, while field sweep is adequate when no obstacles are present. The reason is that the randomization allows to discover a wider variety of search spaces, hence, it can find paths that better outline the obstacles.

6.5.2 Elevated scenario

This test was made in order to analyze the impact of elevation costs in the overall cost. Therefore, the elevation weight (kh) will be iterated across multiple values in different tries. The angle weight (ka) however is set to 0, for now. From Figure 6.9 we can see the elevation of the terrain by simply looking at the shadows of the field, that darker shadow shows us that there is a higher zone at the bottom. Using the interface to fit the area onto the field, generate the tiles, set the start and finish point, and obtain the elevation data, one can now tell that there is a hill with maximum height of 237.7 meters and a lower point at the top right corner with 228.5 meters as shown in Figure 6.10.

(a) Field (b) Area (c) Tiles

Figure 6.9: The second scenario. The starting point is at the top left corner and the finish point in the bottom right.

231.9	231.6	231.4	230.9	230.5	229.9	229.5	228.9	228.5
232.3	232.1	231.9	231.3	230.9	230.4	229.9	229.3	228.9
232.7	232.5	232.3	231.9	231.3	230.8	230.4	229.8	229.4
233.2	232.9	232.7	232.3	231.9	231.3	230.8	230.2	229.8
233.6	233.4	233.2	232.8	232.3	231.7	231.3	230.7	230.2
234	233.8	233.6	233.3	232.8	232.2	231.7	231.1	230.6
234.4	234.2	234	233.8	233.3	232.7	232.2	231.5	231.1
234.8	234.6	234.5	234.3	233.7	233.1	232.7	232.1	231.5
235.3	235	234.9	234.7	234.3	233.7	233.1	232.5	232
235.7	235.5	235.3	235.1	234.8	234.2	233.7	233.1	232.5
236.1	235.9	235.8	235.7	235.5	234.7	234.3	233.7	233
236.7	236.5	236.3	236.2	236	235.4	234.8	234.2	233.6
237.1	237	236.9	236.8	236.5	236	235.4	234.9	234.1
237.7	237.5	237.4	237.3	237.2	236.7	236	235.4	234.8

Figure 6.10: The elevation of the second scenario.

The results are similar to what was done in the benchmark test to observe the impact of the angle cost, but this time we want to see the impact of the elevation costs in the overall cost, hence, the results in Figure 6.11 shows us exactly that. We can see that the higher the *kh* value, the higher the impact of the elevation cost in the overall cost. Additionally, the graph from 6.12 shows us that the drone will climb less with higher values of *kh*, which ends up being beneficial for the power consumption.

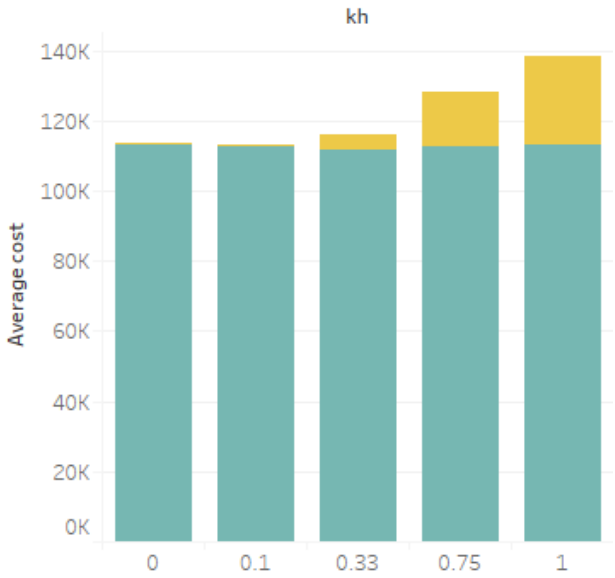


Figure 6.11: The penalty of elevation (yellow) added to the cost of distances (blue) for different penalty weights.

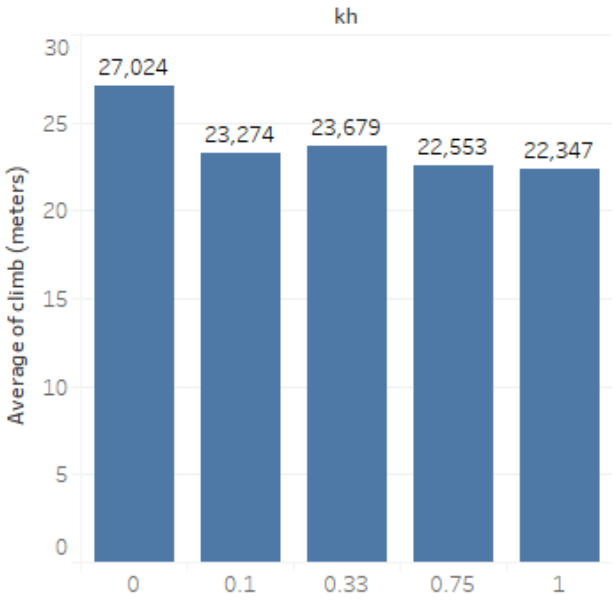
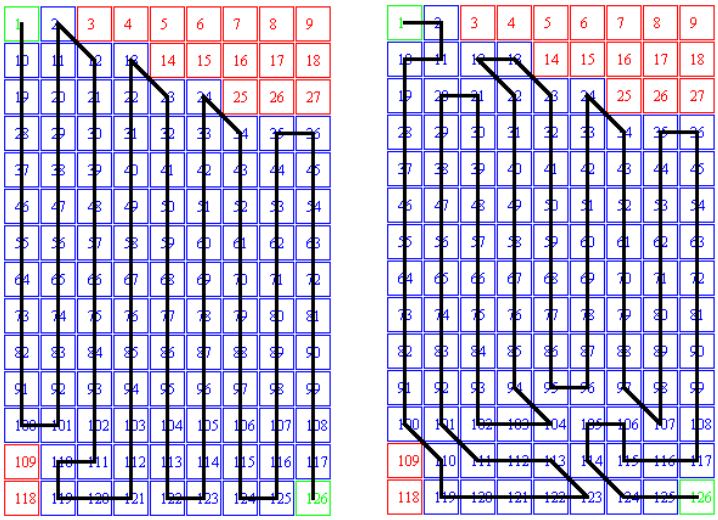


Figure 6.12: The average number of climbs (meters) for different values of *kh*

We also included the weight of the angles by performing a small test and analyzing the impact in the results while comparing two solutions for this field. Figure 6.13 displays two solutions and table 6.7 shows us the configuration of parameters that were set when those solutions were obtained. In solution 6.13b, we can see that, by increasing the weight of the elevation costs to 0.75, three things happen: First, the climb sum (cs) decreases from 26.76 meters to 24.66 meters which is a good sign. Secondly, the sum of turns taken (tt) have increased from 1620 to 2610 degrees. Lastly, we can see in the path itself (Figure 6.13b) that it tries to sweep or cover more cells whenever it passes through the bottom area where the elevations are higher, this is also a good sign because the drone should cover the higher elevations first but at the same time without causing too much impact in the turns. The start and finishing point also play a significant role in this because, ideally, the drone should start from the highest point. However, for whatever reason it may be, the user is free to choose any other point. As mentioned, the interface aids the user in this choice by displaying the elevations of the terrain (Section 4.2.1). In short, it ends up being a gamble between the elevation weight and the angle weight and the final decision should be based on what is trying to be spared the most. If the drone drains more energy when climbing than performing turns, then more weight should be given to the elevation costs.



(a) (b)

Figure 6.13: Two distinct solutions for the second scenario.

sol	cost	h (hval)	ir	*ka*	*kh*	dc	hc	ac	tt	cs
a	120743	vns (0.75)	sweep	0.33	0.33	109243	4701	6805	1620	26.76
b	140498	vns (0.75)	sweep	0.33	0.75	111728	16953	11812	2610	24.66

Table 6.7: The configuration of parameters that were set when the different solutions were obtained. The last five columns stand for distance cost (dc), elevation/height cost (hc), angle cost (ac), the sum of turns taken (in degrees) (tt), and climb sum (cs), respectively.

6.5.3 Obstacle scenario

This last scenario was tested to observe how the system performs when fields with multiple obstacles are used. This scenario is also slightly larger than the one used in the benchmark test. As done throughout the thesis, this system considers the fact that the user may wish to start and end the coverage of the field in different points. This is useful since the finishing cell serves as a transition point for other fields that the user intends to cover. Figure 6.14 shows us an example of what was just described: A road can be seen crossing through the field. We can imagine that the user wishes to start at the beginning of that road, and finish at the end of it, as it is defined in the sub-figure 6.14c. Also, all the obstacles are artificial and were placed there only for testing.

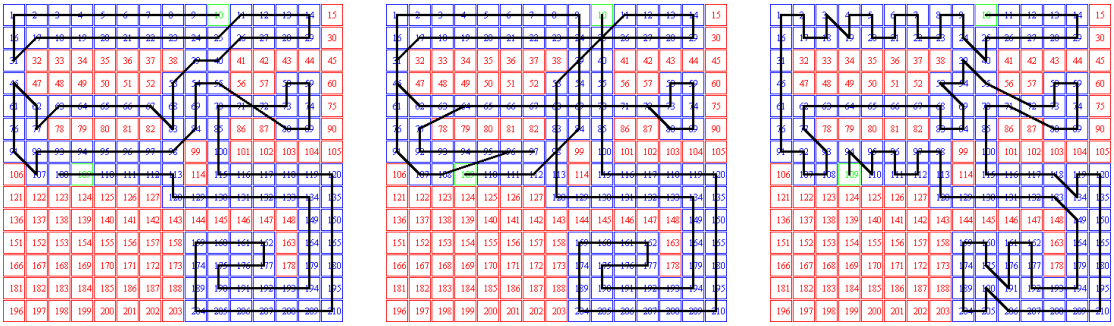
Three distinct solutions are displayed in Figure 6.16. The first was considered to be the most visually appealing, the second solution is the one with less turns made in the path, and the third the solution with less climbs. Table 6.8 shows us the configuration of parameters that were set when these solutions were obtained. Both solutions in sub-figures 6.16b and 6.16c do look strange, but they do what was intended based on its configuration, which is either to reduce the turns or minimize the climbs in the path, respectively. For example, solution 6.16c has reduced its climb by 4 meters when

(a) Field (b) Area (c) Tiles

Figure 6.14: The third scenario. The starting point is the cell from the top and the finish point the one from the bottom.

189.2	189.4	189.6	189.9	190.1	190.3	190.5	190.8	191	191.2	191.4	191.6	191.8	192	192.2
189	189.3	189.5	189.7	190	190.2	190.5	190.7	190.9	191.1	191.3	191.6	191.7	192	192.1
188.9	189.1	189.4	189.6	189.9	190.1	190.3	190.6	190.8	191	191.2	191.4	191.7	191.9	192.1
188.7	189	189.3	189.5	189.8	190	190.2	190.5	190.7	190.9	191.2	191.4	191.6	191.8	192
188.7	189	189.2	189.4	189.7	189.9	190.2	190.4	190.6	190.9	191.1	191.3	191.5	191.7	192
188.6	188.8	189.1	189.3	189.6	189.8	190.1	190.3	190.5	190.8	191	191.2	191.5	191.7	191.9
188.4	188.7	188.9	189.3	189.5	189.7	190	190.2	190.5	190.7	190.9	191.2	191.4	191.7	191.9
188.3	188.6	188.9	189.1	189.4	189.6	189.9	190.2	190.4	190.7	190.9	191.2	191.4	191.6	191.8
188.2	188.5	188.8	189	189.3	189.6	189.8	190.1	190.3	190.6	190.8	191.1	191.3	191.5	191.8
188.1	188.4	188.7	189	189.2	189.5	189.8	190	190.3	190.5	190.8	191	191.3	191.5	191.7
188	188.3	188.6	188.9	189.2	189.4	189.6	190	190.2	190.5	190.7	190.9	191.2	191.5	191.7
187.9	188.2	188.5	188.8	189.1	189.4	189.6	189.9	190.2	190.4	190.7	190.9	191.2	191.4	191.7
187.7	188.1	188.4	188.7	189	189.3	189.6	189.8	190.1	190.4	190.6	190.9	191.1	191.4	191.7
187.6	187.9	188.2	188.5	188.8	189.1	189.4	189.8	190	190.3	190.6	190.8	191.1	191.4	191.6

Figure 6.15: The elevation of the third scenario.



(a) Best visual (b) Best turns (c) Best climbs

Figure 6.16: Three distinct solutions for the third scenario.

compared to solution 6.16b. On the other hand, solution 6.16b has less 3600 degrees in turns. Also, in solution 6.16b, even though both weights are close in value, the altitude cost impact (hc) is much less than the angle cost (ac) in the overall cost. This may be due to the fact that the terrain is not that steep. Observing again the elevation of the third scenario in Figure 6.15, we can confirm that this is true. The difference between the highest point and the lowest point is only 4.6 meters, and if we consider only the flyable cells as we should then it's even lower, 3.6 meters. This value would be the ideal total climb in the path, but of course the drone must work its way around multiple obstacles and as such will make slightly more than that. Sub-figure 6.16a is considered to be the most attractive path for its visual, and by checking the configuration we can see that it stands somewhere in between both solutions 6.16b and 6.16c when it comes to turns taken and climbs. Again, the final decision of which solution to take should be based on what it's trying to be spared the most.

sol	cost	h (hval)	ir	*ka*	*kh*	dc	hc	ac	tt	cs
a	138339	vns (0.75)	rand	0.33	0	122556	0	15785	3690	10.27
b	181220	vns (0.75)	rand	1	0.75	142556	198	38470	2790	12.16
c	120987	vns (0.75)	sweep	0	0.33	121971	-977	0	6390	8.17

Table 6.8: The configuration of parameters that were set when the different solutions were obtained. The last five columns stand for distance cost (dc), elevation/height cost (hc), angle cost (ac), the sum of turns taken (in degrees) (tt), and climb sum (cs), respectively.

Chapter 7

Conclusion

Throughout this dissertation we've seen that the implemented system assists in the planning of a farm field coverage using a drone and optimizes its path considering a set of factors. One of these factors is not often employed in todays related research, which is the elevation factor of the terrain. The other factors include the possible obstacles across the field, distinct start/finish positions and the amplitude of turns. After modeling the drone optimization problem and respective factors, we've succeed in reducing both the climbs and the amplitude of the curves made throughout the path.

Additionally, the interface grants the user the ability to choose any desired field from actual crops found anywhere in the world. This process includes the selection of the farm field, the obstacles and the start/finishing positions. The generated instance then contains all the necessary information to further find and optimize a coverage path, including the elevation of the terrain. The interface also informs the user about restricted areas and legal actions that must be followed when operating the drone. Apart from the main components that were just described, a debugger/visualizer was also included in this system to aid the process of optimization.

The first stage of testing allowed us to set VNS as the preferred metaheuristic for this drone route optimization problem using 75% of the total number of cells as the maximum neighborhood window (Section 3.2.2).

The second stage of testing consisted in comparing this system with the one selected as a benchmark [9] using a configuration of parameters that resembles their system. The results have proven to be consistent and effective in achieving good solutions.

The third and final stage of testing had the purpose to use the interface to gather actual fields from anywhere in the world and to study the visual appearance of the paths and its corresponding costs. We've seen from the results of this test that the system is capable of providing good solutions for different types of fields, from either the most common

ones, which are flat crops with no obstacles in the middle, to more complex fields with multiple obstacles and steepness. We've also concluded that the randomization heuristic for the initial route is more adequate for fields with multiple obstacles, while the field sweep method is suitable when there are none.

These tests also led to the conclusion that by tweaking the weights of both elevation and angle costs we can get different solutions suitable for what is trying to be achieved. For example, by slightly increasing the elevation weight costs the path started to cover more cells around an elevated area before going to lower grounds and without causing too much impact in the turns. The same applies to the weight of angles, since a minimization of the amplitude of turns were shown as a result. This process of finding the suitable solution for what is trying to be spared in terms of cost is also a big challenge on its own. For example, if the drone has a really heavy tank of chemicals to spray over the crop, then it would make sense to increase kh in order to minimize the climbs. Or if for some reason, the drone's consumption model specifies that the drone drains much more energy when performing high amplitude turns, then perhaps we should increase the ka. This process was left out as a point of research as will be mentioned in the Future work section.

7.0.1 Future work

Tests dimensionality

The dimensionality of parameters that could be tested is excessive. Both metaheuristic and system parameters needed to be considered. The SA metaheuristic alone has multiple parameters to be tuned. Hence, the reasonable approach to this problem was to narrow down the possible parameters by choosing the main parameter for each metaheuristic, as it was done in this dissertation. However, the ideal would be to further tweak the vast number of paramaters of SA.

Drone's consumption model

As mentioned, the process of finding a suitable solution based on a drone's consumption model is a challenge on its own. The planner as it is now, gives us a generalist idea of how the system performs by using raw cost functions. Therefore the proposal is to tweak the system's parameters in such way that it finds the most suitable solution for a specific farming drone. Additionally, we could also take into account the fact that the weight of the tank (full of chemicals) decreases over time throughout the coverage of the field. Other operation conditions such as the wind may also be considered in a more detailed model.

Area subdivision

As described in Section 5.4, one of the biggest obstacles in these type of problems is the dimension of field that is being considered. The research chosen as benchmark [9] used a reasonable size field (Figure 2.3), and as such, the tests on this dissertation used similar fields in terms of dimension, some bigger, others smaller. However it would also be valuable to include a technique to allow the use of larger instances. The area subdivision technique was not fully developed, it only takes care of subdividing the field into multiple, smaller, fields. The harder part, which was not developed, is to find a transition point between all these smaller fields and then integrate this search technique in the current system.

Components integration

The system currently has three components. The planner, the interface and the visualizer, which may be considered as part of the planner. The ideal user-interaction with this system would be to click a button in the interface and automatically call the planner to start optimizing the solution. The interface would be constantly updating the display, showing the best solution found until some moment in time. Then, after some elapsed time (defined by the user), the planner would stop optimizing and show different types of solutions, where for example one solution would have minimal climbs while other minimal turns, giving the user the freedom to choose between a set of solutions. This fully-integrated system was not implemented. Most of the system is automated, but the interface was developed using Javascript, hence, it is not allowed to access local executable files such as starting the planner. A server would be ideal to call the planner and keep receiving its process information.

Bibliography

[1] A. N. da Aviação Civil. *Código Drone*. July 2016. URL: https://www.voanaboa.pt/codigo-drone.

[2] A. N. da Aviação Civil. *Regulamento relativo às condições de operação aplicáveis à utilização do espaço aéreo pelos sistemas de aeronaves civis pilotadas remotamente ("Drones")*. July 2016. URL: https://www.voanaboa.pt/regulamento.

[3] C. D. Franco and G. Buttazzo. "Energy-Aware Coverage Path Planning of UAVs." In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*. 2015, pp. 111–117. DOI: 10.1109/ICARSC.2015.17.

[4] E. Galceran and M. Carreras. "A Survey on Coverage Path Planning for Robotics." In: *Robot. Auton. Syst.* 61.12 (Dec. 2013), pp. 1258–1276. ISSN: 0921-8890. DOI: 10.1016/j.robot.2013.09.004. URL: http://dx.doi.org/10.1016/j.robot.2013.09.004.

[5] J. Jin and L. Tang. "Coverage path planning on three-dimensional terrain for arable farming." In: *Journal of Field Robotics* 28.3 (), pp. 424–440. DOI: 10.1002/rob.20388. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20388. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20388.

[6] Lee. *Simulated Annealing for beginners*. Apr. 2013. URL: http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6.

[7] M. Mazur. *Six Ways Drones Are Revolutionizing Agriculture*. July 2016. URL: https://www.technologyreview.com/s/601935/six-ways-drones-are-revolutionizing-agriculture/.

[8] J. Miller, S. Kim, and T. Menard. "Intelligent Transportation Systems Traveling Salesman Problem (ITS-TSP) - a specialized tsp with dynamic edge weights and intermediate cities." In: *13th International IEEE Conference on Intelligent Transportation Systems*. 2010, pp. 992–997. DOI: 10.1109/ITSC.2010.5625106.

[9] J. Modares, F. Ghanei, N. Mastronarde, and K. Dantu. "UB-ANC planner: Energy efficient coverage path planning with multiple drones." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 6182–6189. DOI: 10.1109/ICRA.2017.7989732.

[10] L. H. Nam, L. Huang, X. J. Li, and J. F. Xu. "An approach for coverage path planning for UAVs." In: *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*. 2016, pp. 411–416. DOI: 10.1109/AMC.2016.7496385.

[11] M. Torres, D. A. Pelta, J. L. Verdegay, and J. C. Torres. "Coverage path planning with unmanned aerial vehicles for 3D terrain reconstruction." In: *Expert Systems with Applications* 55 (2016), pp. 441 –451. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.02.007. URL: http://www.sciencedirect.com/science/article/pii/S0957417416300306.

[12] Wikipedia. *2-opt*. Aug. 2018. URL: https://en.wikipedia.org/wiki/2-opt.

[13] Wikipedia. *Comet (programming language)*. May 2018. URL: https://en.wikipedia.org/wiki/Comet_(programming_language).

[14] Wikipedia. *Floyd–Warshall algorithm*. Aug. 2018. URL: https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm.