UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Design of an efficient path planning and target assignment system for robotic swarms in agricultural applications

Jochem Postmes

M.Sc. Thesis
April 2021

Supervisors:
prof. dr. P. J. M. Havinga
dr. A. Kamilaris
dr. M. Poel

Pervasive Systems Group
Department of Computer Science
Zilverling Building
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Preface

Dear reader, thank you for taking the time to have a look at my master thesis. The process has been more challenging than I anticipated, not in the least because of the global pandemic. Never having physically met my supervisors and a lack of physically exchanging ideas with my peers have made the process more solitary than expected. Nonetheless, I am proud of the end result that lies before you.

Making agriculture more sustainable and efficient is highly relevant at the moment, with global warming, climate change and an ever growing world population. It was not hard for me to see the importance of contributing something to this process, and I am glad I got the chance to finish my studies at the University of Twente within this topic. It is wonderful to see that advances in this field of technology might be the solution to several of the major problems we are currently facing.

I would like a to thank a few people who either contributed to this project, or helped me in the process. First off, I would like to thank my main supervisor, Andreas Kamilaris, for being a continuous source of inspiration. Moreover, I would like to thank him for his patience with my many questions and for taking the time to review my work, despite being very busy. Secondly, I would like to thank Nicolò Botteghi for all his unconditional support and help with technical issues. His expertise in the field of reinforcement learning proved to be a valuable asset. Thirdly, I would like to thank Beril Sırmaçek for reviewing my work and providing valuable feedback. Lastly, I would like to thank my parents and Sascha, for providing mental support whenever I needed it and for sparring with me about ideas and concepts I was struggling with.

Please enjoy this account of my work over the past 8 months. I hope this thesis will inspire you and be the motivation for future work in this topic!

Summary

Automation in agriculture is a growing topic in operations research. Deploying a swarm of robots to perform precise agricultural tasks is a possible solution to various problems, such as soil compaction and decreasing biodiversity. However, this solution brings along some challenges. Path planning and target assignment is a computationally complex problem, which becomes increasingly difficult for larger numbers of robots or large agricultural topologies. This research thesis continues a previous project from the University of Twente, in which a first step was taken in solving the challenges of path planning for robotic swarms. In this iteration of research into the subject, a broader range of algorithms is investigated. Moreover, several of these algorithms are compared against each other to find the most suitable approach to different kinds of tasks. Lastly, a step is taken towards a real-world implementation of a path planning system. In general, this research project is a design problem; it aims to create a system with the best possible approach to path planning and target assignments for robotic swarms in agricultural fields.

As a starting point, the problem and previous work is described in more detail, after which the problem is identified as a kind of Vehicle Routing Problem (VRP). Some agricultural tasks can also be seen as Coverage Path Planning (CPP).

This project includes a state of the art research, in which a range of algorithms is explored. A number of approaches to solve the VRP are discussed, including exact methods, heuristic methods and metaheuristic methods. Moreover, some research is done into Multi-Agent Reinforcement Learning (MARL), to see if it is a viable solution to the path planning problem. Some other approaches are also discussed, including a few other reinforcement learning and deep learning approaches.

The algorithms from the state of the art research are compared against each other in a design space exploration in order to make a selection. A number of specifications for the path planning system are provided, after which the assumptions under which the system operates are discussed.

An interface to import real-world topologies from Google Earth into a simulation environment is used to create ten different representations of agricultural tasks. Moreover, a setup is created to test how well methods scale against increasing numbers of agents (robots) and landmarks (points of interest).

Each of the selected algorithms is used to produce a path planning solution for each of the ten tasks. The resulting solution is given a score (cost), and the computation time necessary to produce the solution is recorded as well.

The obtained results show how well each algorithm scales, as well as how well each algorithm performs on the different tasks. There appears to be no clear difference between the separate tasks, only in problem size. The only limiting factor is the computation time of the algorithms, as expected. Two possible heuristic algorithms, Clarke-Wright and Christofides produce fast solutions, that are solid but can be improved upon. Christofides, the best-performing of the two, is thus most suitable for larger problems or problems that change during operation (and thus needing recalculations). For smaller problems, more robust solutions can be obtained using one of the metaheuristics available in Google's OR-Tools toolkit. These have higher computation times, but produce better solutions. Using them is thus infeasible for larger problems, and when not enough computation time is available for the problem. The best-performing of the available metaheuristics is Tabu Search.

The final system selects which of these algorithms to use for path planning based on the problem size, number of available agents and possibly other factors. Real-world GPS data can be used as an input using Google Earth, and the system returns its solution in GPS coordinates as well, ready to be used by robots in the field. Moreover, a Graphical User Interface (GUI) is made to allow accessible use of the system and to easily recreate the experiments described in this thesis. Some recommendations for improving the system and continuing the research are given as well.

Contents

List of acronyms

GPS	Global Positioning System
UAV	Unmanned Aerial Vehicle
GUI	Graphical User Interface
CPP	Coverage Path Planning
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
CVRP	Capacitated Vehicle Routing Problem
SMT	Satisfiability Modulo Theory
NN	Nearest Neighbour
LKH	Lin-Kernighan-Helsgaun
CW	Clarke-Wright
SA	Simulated Annealing
TS	Tabu Search
GLS	Guided Local Search
GA	Genetic Algorithm
ACO	Ant Colony Optimization
PSO	Particle Swarm Optimization
RL	Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process

DQN	Deep Q Network
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MAAC	Multiple Actor-Attention Critic
COMA	Counterfactual Multi-Agent
MFQ	Mean Field Q-Learning
NLP	Natural Language Processing
LUT	Lookup Table

Chapter 1

Introduction

1.1 Motivation

Since global population has grown significantly over the past few decades, and is projected to grow by another 40% in the coming 50 years [1] (see Figure 1.1), the demand for food keeps growing. Efficient and responsible farming can be the solution to this problem, and agricultural automation plays a large role in this. Agriculture is one of the sectors which have seen the most automation in the previous century, mainly in the form of large field machinery and in dairy production [2]. The increased automation means a decrease in the amount of human labour necessary on farms, and thus a single farmer can cultivate a larger area. However, current farming approaches are largely monoculture, which leads to soil exhaustion and a higher change of pests and crop disease [3]. Moreover, the large and heavy machinery used causes soil compaction, restricting natural water and nutrient cycles [4]. Thus, automation in agriculture will need to adapt to more biodiversity in fields and more lightweight machinery [5].

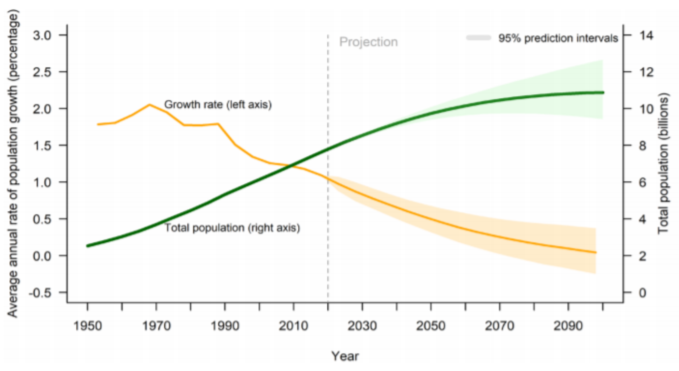


Figure 1.1: Population size and annual growth rate for the world: estimates for 1950-2020, and medium-variant projection with 95 percent prediction intervals for 2020-2100. United Nations Population Facts [1]

Nowadays, automation comes mainly in the forms of robotics and artificial intelligence. The advances in technology have made precision agriculture a possibility, with Global Positioning System (GPS) and satellite images playing the lead role [6]. Precision agriculture increases crop efficiency, for example through monitoring vegetation health and soil quality. Robots deployed in precision agriculture are often small robots, such as drones, which have a shorter battery life than their larger counterparts. If a group of smaller robots cooperates in order to cover a larger area of ground, their short battery life becomes less of a problem, as the work can be divided such that each robot is working for approximately the same amount of time. This will also distribute maintenance needs more evenly. Moreover, cooperation can lead to better and more efficient coverage of a field, as well as the possibility for deploying different robots with different tasks in more diverse fields [7].

Notably, deploying robots for agricultural tasks will mean a decrease in expenses for the farmer as well; especially since autonomous robots are becoming cheaper and cheaper. The purchase of one or more industrial drones combined with their maintenance and electricity costs, outweighs the cost of manual labor combined with the machinery used for large-scale monoculture farming [8].

Thus, deploying robots in swarms for precision agriculture could be the solution to making agriculture more efficient and sustainable. However, path planning and task assignment for the robots in a swarm is a difficult topic. There are several algorithms and approaches that solve similar problems, but it is unclear which of these perform optimally under which circumstances. Furthermore, the speed and computational complexity of these methods may vary based on the environmental constraints and the number of robots involved. Lastly, this approach can also be extended to non-agricultural applications, such as monitoring wildlife or spotting forest fires.

1.2 Clarification of terms

Before the project and its goals are defined, some of the terminology used in this report is defined. This section aims to clarify the most important terms used, for a more understandable reading experience of the thesis and to avoid any misconceptions on what the terms represent.

Agents

This research investigates path planning for agricultural robots in a group (also referred to as *swarm*). Individuals in such a group are referred to as *agents*. Agents represent either ground-based robots or aerial robots (*drones* or *UAVs*). Depending on the application, agents might also be referred to as *vehicles*.

Landmarks, spawns, routes

In precision agriculture or field monitoring, an agent can be tasked with visiting a number of locations (*points of interest*). These points either represent a spread across a field such that the agents cover the entire field, or single points of interest, for example sick crops that need treatment. These points will often be referred to as *landmarks*, *nodes*, or *customers* in some specific situations.

A special type of landmark is the point where agents start and end their tasks. This point is referred to in various ways. Depending on the application, it might be referred to as the *agent origin*, *spawn*, *depot* or *nest*.

A list of landmarks that an agent should visit sequentially is referred to as that agent's *route*, *tour* or *path*. A group of routes, one for each agent, is a *solution* to the problem at hand.

Topology, Scenario, Task

A *topology* is a representation of a real-world agricultural field. A topology has a *boundary*, which can normally not be crossed by the agents. A topology with a number of landmarks and an agent origin represents a planning *task* - a *scenario*. The scenario and the topology together form the *environment* in which the planning tasks will be simulated.

Approach, System

To formulate a solution to the routing problem in an environment, a specific *approach* will be used. An approach consists of an algorithm, or combination of algorithms, which transforms the information from the environment into a solution. The total application which takes the environment data as input and produces a viable solution as output is referred to here as the *system*.

1.3 The project

This research project aims to combine existing methods in such a way that the problem described in Section 1.1 can be solved for various practical scenarios. This thesis is the continuation of a previous research project from the Robotics and Mechatronics group at the University of Twente [9]. The work on this topic is continued, taking previous conclusions into account. A broader spectrum of algorithms is investigated, and compared against the already tested ones. Furthermore, the simulation scenarios are expanded in two main ways: the number of agents is increased and field topologies are made more complex, for example by increasing the number of landmarks, adding different kinds of obstacles and using irregular shapes of fields. Lastly, another step towards practical implementation of a path planning system is made; by using GPS coordinates of actual fields, by creating a Graphical User Interface (GUI) for accessible use and by producing sets of target GPS coordinates for the robots.

1.4 Goal

This research project has two main goals:

- Optimization of a path planning and target assignment system for robotic swarms in agriculture
- Working towards a real-world application for the system

Combining these goals leads to the main research question:

> ***How to design a system for efficient path planning and target assignment for robotic swarms in agricultural applications?***

This main question will be answered by finding an answer to the following, more specific questions:

- *How can be ensured that the system scales well with regard to the number of agents in the swarm?*
- *Which approach is best suited for which individual topologies and/or combinations of topologies?*
- *How to simulate a realistic scenario of a robotic swarm in an agricultural field, taking into account the robots' constraints and limitations?*

1.5 Thesis structure

This thesis is structured as follows. First, the background of the project is discussed in Chapter 2, whereas the state of the art can be found in Chapter 3. In Chapter 4, the requirements and assumptions of the project are specified. Next, the simulation and test specifics, and the approach for answering the research questions can be found in Chapters 5 and 6, respectively. In Chapter 7 the results of the simulations and testing are provided, and in Chapter 8 these results and the general approach are discussed and evaluated. Finally, in Chapter 9 conclusions are drawn and recommendations are done for future work.

Chapter 2

Background

In this chapter, some background is provided on the research project. Firstly, the previous work on the project is described. Next, some background is provided on the problem and different uses for robotics in agriculture. Lastly, the following path planning problems are explained: coverage path planning, the traveling salesman problem and the vehicle routing problem.

2.1 Previous work

In the previous iteration of this project [9], the problem of path planning for a robotic swarm is transformed to a Vehicle Routing Problem (VRP) - partially as an extension to Coverage Path Planning (CPP). Both of these concepts are explained further in Sections 2.3 and 2.4. Describing the problem as a VRP is done in this thesis as well.

To solve the path planning and target assignment problem, a few promising algorithms were investigated through the use of simulations in Python. This includes the testing of various simple scenarios and topologies, as well as a distinction between ground-based and aerial robots. Scenarios with up to five agents were considered.

The algorithms that yielded the best results were Christofides' algorithm and Ant Colony Optimization (ACO). Christofides' algorithm is used in combination with a clustering algorithm to extend it to a multi-agent approach - the method by itself is not suitable for solving a VRP. The clustering method with the best results is k-means clustering. Both ACO and Christofides' algorithm are considered in the rest of this research and compared to other discussed methods (see also Section 3.1). Moreover, the system is expanded to account for larger numbers of agents, as well as more complex topologies. Also, some of the previously made assumptions are relaxed whenever possible. Lastly, this thesis takes another step towards practically implementing a path planning system for agricultural applications.

2.2 Drones & ground robots in precision agriculture

Aerial robots in precision agriculture can be deployed to serve various means [10] [11]. The most common occurrence is in crop monitoring, where a drone or group of drones surveys a field to identify crop conditions, such as their health, color and density. At the same time, the field itself can be monitored to observe soil quality and elevation. In general, the effect of climate change can be tracked, as well as diseases and the emergence of new types of bugs. However, monitoring is not the only application for these types of drones. From the air, a Unmanned Aerial Vehicle (UAV) can fulfill seeding and planting tasks. It can also be used for small irrigation tasks and the spraying of pesticides.

Ground-based robots are harder to use for global crop monitoring. However, they can be deployed for more precision-based work, such as crop treatment and harvesting. Other tasks for these robots can include seeding, planting, weeding, irrigation and fertilization. An advantage of ground robots over UAVs is that they usually have a longer battery life, since they are less weight-restricted. However, they are restricted in other ways, such as their slower movement speed and possible obstacles on the ground.

2.3 Coverage Path Planning

In coverage path planning, the goal is to explore an area in such a way, that all relevant locations are mapped or visited [12]. For example, this can be relevant in aerial crop monitoring: in order to do this, images have to be taken of all crops in the field. Using CPP to approach this, means that drones fly a path over the field such that their camera is able to capture all of the necessary data.

There are various approaches for solving the CPP problem [13], including describing it as a type of vehicle routing problem. Other examples include initializing geometric patterns on the field, and finding the paths from there. A recent project, which is explained in depth in Section 3.3, solved the problem in a different way - by deploying a Satisfiability Modulo Theory (SMT) solver [14]. Furthermore, there exist deep learning approaches, which are also discussed in Section 3.3.

2.4 Vehicle Routing Problems

Generally, the problem of path planning for a group of robots can be described as a type of vehicle routing problem (VRP) to come to an efficient solution [15]. CPP can also be formulated as a type of VRP. VRPs are a generalization of the Traveling

Salesman Problem (TSP), which is elaborated in the next section. Finding an efficient solution to the VRP has been a highly relevant topic ever since it was first formulated in 1959 by Dantzig and Ramser [16]. A large number of possible solutions have been proposed thus far, of which a few promising ones are discussed in Section 3.1.

Traveling Salesman Problem

The Traveling Salesman Problem can be formulated as follows: given a set of cities, what is the shortest path through all the cities - without visiting any city more than once. This is an NP-hard problem, meaning that finding an optimal solution becomes increasingly difficult if the number of cities increases. There exist two types of TSPs: the symmetric and asymmetric case, where in the symmetric case the distance between two points is the same in both directions (undirected), and in the asymmetric TSP this is not the case. For this research, the focus is on the symmetric TSP, since most solving methods assume the problem to be symmetric. However, the asymmetric case might be relevant as well, for example if slopes (for ground robots) or wind direction (for aerial robots) have to be taken into account.

Any VRP is an extension of the TSP, and is therefore also NP-hard. To circumvent the complexity problem, solutions often use a set of heuristics to approximate an optimal solution instead.

Types of Vehicle Routing Problems

Over the years, various types of VRPs have been specified to describe different problems. The most common one is the Capacitated Vehicle Routing Problem (CVRP), which assumes a single starting depot for all vehicles, and a maximum carrying weight capacity for each vehicle. Another common type is the VRP with time windows, which specifies times at which locations can be visited by the vehicle. The problem can be extended to include pick-up at multiple locations, backhauls and multiple depots.

All of these could be translated into the case of multiple robots: when each location is a point of interest on a field to be visited, the robot's carrying capacity could be relevant when it has to carry pesticide. Harvest tasks can be viewed as a pick-up and delivery system. On a large-scale farm, the agents might be stored in multiple depots to reduce travel times to their location of operation. Lastly, some tasks might require a specific time at which they need to be performed, making the problem related to the VRP with time windows.

2.5 Background conclusion

In this chapter, the background on the project has been described. The most straightforward way to formulate the problem at hand is to transform it into a type of vehicle routing problem. However, this is not the only possible approach. In the next chapter, a wide range of algorithms are discussed; both approaches for solving the vehicle routing problem, and other approaches as well.

Chapter 3

State of the Art

This section outlines the state of the art in solving path planning and target assignment problem. Firstly, several traditional path planning approaches are discussed, divided in exact methods, heuristic methods and metaheuristic methods. Next, the possibility for applying multi-agent reinforcement learning to this problem is discussed. Lastly, some other noteworthy approaches are mentioned, several of which are based on reinforcement learning and deep learning. In Chapter 4, the information on these approaches is compared to draw conclusions on which methods are most suitable to start implementing and testing.

3.1 Traditional path planning approaches

There are various approaches to solving VRP-related problems, many of which are more general combinatorial optimization methods. In this section, various path planning and optimization algorithms are explored. The algorithms are split into three subcategories: exact algorithms, heuristic algorithms and metaheuristic algorithms.

3.1.1 Exact algorithms

There exist various algorithms that can calculate an optimal path for the basic TSP, including dynamic programming and iterative linear programming approaches. However, these algorithms have a high complexity and scale dramatically when the number of points increases, due to the problem being NP-hard, as described earlier. For example, the most straightforward algorithm to give an exact solution (sometimes referred to as brute-forcing the solution) calculates every possible path and finds the best one, with a substantial complexity of $O(n!)$. There are faster approaches available, of which Concorde [17] is often regarded as the fastest. Because of their complexity, exact algorithms are infeasible solutions to the problem and are not considered any further in this research. However, an exact solver like Concorde can be useful for verifying whether a solution is indeed the optimal one, which has been done for VRP solutions already [18].

3.1.2 Heuristic algorithms

Heuristic algorithms aim to reduce the complexity of finding a VRP solution by using a set of heuristics. The solution is found in a much shorter time than the exact solution, at the cost of losing guarantees about the solution being optimal. Some examples of heuristic solutions are shown in Figure 3.1, comparing them to the exact solution.

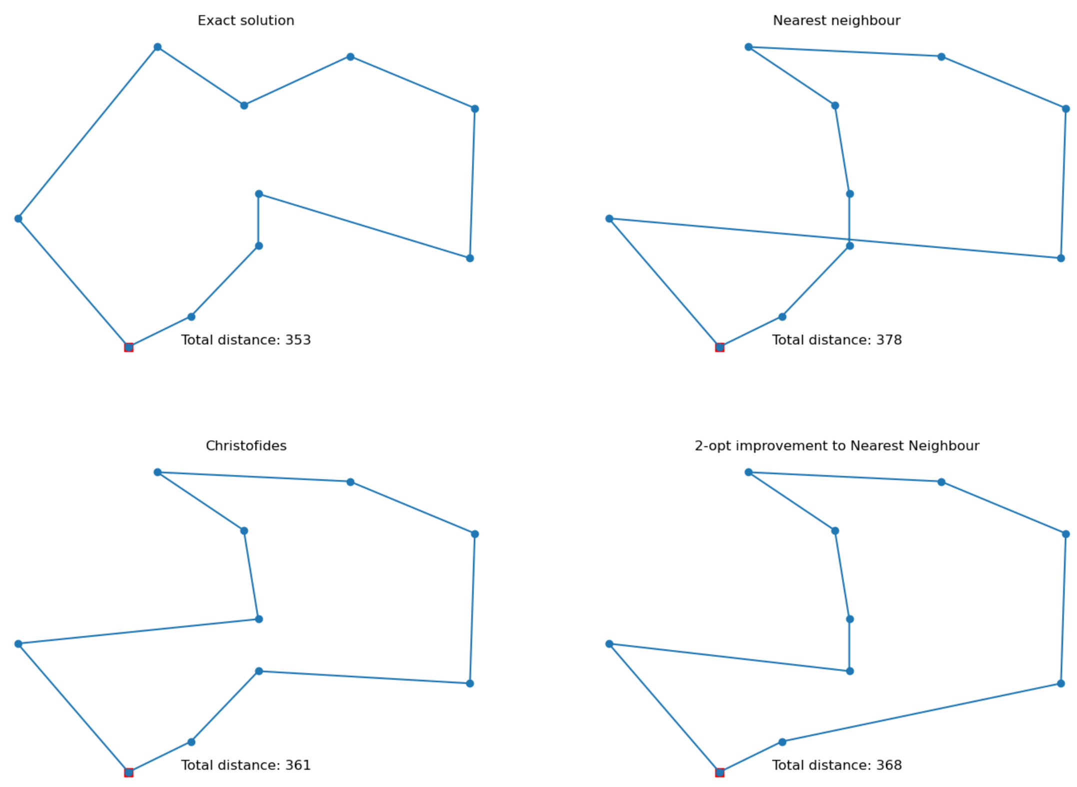


Figure 3.1: Different heuristic approaches (NN, Christofides and 2-opt improvement) compared to the exact solution for a single TSP ($n = 10$).

The Nearest Neighbour heuristic

On of the simplest heuristic approaches to path planning is the Nearest Neighbour (NN) heuristic [19]. The path is planned in a greedy way: by always moving to the closest point that has not been visited yet. The complexity of this algorithm is $O(n^2)$, with n the number of points. This heuristic works fine for small-scale planning, however, for larger planning problems the path is often far from optimal. This is because it only considers one point at a time, often backtracking (see Figure 3.1). The performance of the algorithm can be slightly improved by repeating the search method from each possible starting point, and selecting the best solution from these. The complexity then increases to $O(n^3)$ [20]. The main advantage of this method is its simplicity.

Christofides' algorithm

The best worst-case performance out of the researched heuristics is achieved by Christofides' algorithm [21]. Based on calculating minimum spanning trees between groups of points and Euler tours, the algorithm has a complexity of $O(n * log(n))$. The worst case path length is at most 1.5 times the optimal path length, i.e.:

$$\frac{Path_{Christofides}}{Path_{optimal}} \leq \frac{3}{2}$$

Since this is one of the few available algorithms that gives a worst-case performance bound, it can be used efficiently to validate other methods - especially in combination with its low execution times.

k-opt

The k-opt swapping method [22] [23] is a heuristic often deployed by other approaches to improve path lengths for shorter segments. The heuristic has a complexity of $O(n^k)$. It is a local search method, where in each iteration k edges of the current path are replaced by k new ones such that the total path length decreases. This is repeated until no improvement is found. Deploying k-opt as a main approach is often too computationally expensive, especially for values of $k = 3$ and higher. However, it is used to further optimize the paths in some of the mentioned heuristic and metaheuristic approaches in this chapter. An example of using 2-opt to improve a Nearest Neighbour solution is shown in Figure 3.1.

The Lin-Kernighan-Helsgaun algorithm

Another early effective heuristic for the TSP has been established by Lin and Kernighan [24]. Their method was later adapted into a more successful implementation by Helsgaun [25], often referred to as the Lin-Kernighan-Helsgaun (LKH) algorithm. The algorithm works as follows: an initial path through all points is made using another method, for example NN. Then, a k-opt approach is used, where k is varied in each iteration, based on which value yields the best result. Both the original algorithm and the implementation by Helsgaun have a runtime complexity of approximately $O(n^{2.2})$, slightly worse than the NN heuristic. Often, especially for smaller numbers of cities, LKH is able to find an optimal path either on the first run or within a few runs with other initial paths. Therefore, it has a high average result quality. However, there is no performance guarantee such as with Christofides' algorithm, since the results can vary based on the type of problem.

The Clarke-Wright savings heuristic

The Clarke-Wright (CW) savings heuristic was developed specifically to solve the CVRP, being one of the first approximate solutions to this problem [26] that is still used nowadays. It is a greedy method, which calculates a savings value S_{ij} for each possible combination of points, based on the saved transportation cost by combining the points on a single route. The higher this value, the more attractive it is to include this route in the vehicle routing. An ordered list of point pairs is made, sorted by their savings value. From this list routes are constructed, either sequentially or in parallel, as explained by Lysgaard [27]. The parallel approach often yields better results, at the cost of being more computationally expensive. The routes are only considered valid if the vehicles' capacity is not exceeded. Other VRP methods often use the solution generated by CW as a starting point to improve upon.

3.1.3 Metaheuristic algorithms

Heuristic path planning algorithms use heuristics to generate a solution to optimization problems. Metaheuristic algorithms apply heuristics in another way: instead of generating one specific solution based on heuristics, they use more abstract heuristics to find a heuristic approach which generates sufficiently good solutions to the problem. In this way, metaheuristics make few assumptions about the problem itself and more about the way of finding an acceptable solution, for example by using heuristics to reduce a problem's search space. Six of these approaches are discussed in this section, all of which can be used to find a solution to the VRP.

Simulated Annealing

Simulated Annealing (SA) is a probabilistic global optimization method, which arose as an analogy to annealing; the natural principle of heating and cooling a material to alter its physical properties. It has proven to be a successful method to approximate difficult combinatorial problems, such as the TSP [28]. The method needs an initial current state (path), and then considers a neighbouring state (for example, the same path but with two cities swapped). For both states, their cost / energy is compared, and if the neighbouring state has a lower cost, it has a possibility of becoming the new current state. Otherwise, the current state stays the same as before. This decision is made based on a probability function, that takes as input the costs of both paths and a time-varying factor called the Temperature (T). T decreases over time, and once it becomes zero, the algorithm simply keeps moving to the state with the lower cost (greedy approach). In this way, the global energy is minimized to approximate an optimal solution while avoiding getting stuck in local minima. Osman [29] shows in his algorithm comparison that SA can be applied to achieve good results on the VRP, however, it is outperformed by Tabu Search in both computation time and solution quality.

Tabu Search

Tabu Search (TS) is a method used for solving combinatorial optimization problems, which operates in a similar manner as SA. It also moves from a possible solution towards an improved possible solution, based on a local search. For each solution, its neighbourhood is evaluated through the use of various memory structures, which combined form the *tabu* list. The best solution in the neighbourhood is selected and becomes the new current best solution. The tabu list outrules the exploration of certain solutions that violate predefined rules or have been visited recently. The memory structures of this tabu list can differ per method, for example using only a short-term memory, a long-term memory, or a combination of both. Similar to SA, Tabu Search can be applied to both the TSP and the VRP. In the work of Osman [29], Tabu Search is the best performing algorithm on the VRP, and the version by Gendreau [30] achieves even a slightly better performance.

Guided Local Search

Guided Local Search (GLS) can be seen as a special case of Tabu Search. It is another metaheuristic method that uses heuristics to guide a local search and avoid local minima [31]. Each solution feature (in this case the order of cities, for example) is given a penalty when a local minimum is found, to guide the search out of it. This approach is repeated iteratively until a stopping criterion is reached, after which the best recorded solution is reported. The method is implemented in Google's OR-Tools [32], together with SA and TS, and is reported to have the best results of the three when it comes to VRPs. However, as with any heuristic-based approach, this may vary depending on the problem.

Genetic Algorithms

A Genetic Algorithm (GA) is a meta-heuristic algorithm based on biological genetics, as the name suggests. It can apply the concepts of evolution and natural selection to optimization problems and search problems. In each iteration, a population of solutions is created. Each of these solutions receives a fitness score, which in the case of a VRP would be based on total path length, for example. For the next iteration, offspring solutions are generated by slightly modifying and/or combining the parent solutions, where parent solutions with a better fitness score are more likely to reproduce. Various GA approaches to the VRP exist, though early versions cannot compete with the Tabu Search and SA approaches mentioned earlier. The implementation of Baker and Ayechew for the basic VRP [33] is able to approach the optimal solution within 0.5% of the best known values on average. In another approach, it is shown that a GA can be used to solve various extensions of the VRP, including the multi-depot variant [34].

Ant Colony Optimization

A different category of algorithms that solve combinatorial problems is found in swarm intelligence, where social structures and interaction of social insects are studied. In ACO, a set of artificial ants are deployed to solve an optimization problem. When applied to a path planning problem, ants start by moving in a semi-random fashion from location to location, depositing pheromone on each path they take. This pheromone increases the probability for other ants to take this path. Pheromone levels decrease over time (evaporate), meaning that if only a small number of ants take a certain path or if traveling this path takes a lot of time, a lot of the pheromone on the path will be gone and the path is thus less desirable. After repeating the process for a number of iterations, a path has been produced for each ant based on the pheromone levels. The approach can be used successfully to find good solutions to the VRP, and with minor extensions to multi-depot problems as well [35]. An improved version is able to compete with the solutions of SA and TS, achieving a high benchmark solution quality [36].

Particle Swarm Optimization

Another approach based on swarm intelligence is Particle Swarm Optimization (PSO). Of the investigated metaheuristic methods, this one has the least literature available on applying it to the VRP. However, there exist approaches that show that PSO can be applied to find proper solutions, for example in a hybrid form combined with a GA [37]. This hybrid approach is able to achieve high benchmark results that are able to compete with other methods successfully. PSO has also been applied to a few specific types of VRPs [38] [39], which might be relevant if the researched problem is to be extended with more constraints. In PSO, a population of particles update their position constantly, based on both their individual best position and the overall best position. Each 'position' here represents a possible solution to the problem being evaluated.

3.2 MARL for path planning and target assignment

A useful tool that is becoming increasingly more popular in robotics and operations research is Reinforcement Learning (RL). This method has been used to tackle the problem of path planning and target assignment already, but often literature only considers a single agent [40]. Moving from single agent learning to Multi-Agent Reinforcement Learning (MARL) is a difficult step in RL - a step which is necessary to model path planning for a swarm of robots in any environment. The step is a difficult one due to the fact that, generally, RL is based on a Markov Decision Process (MDP), which needs a number of assumptions to hold. One of these assumptions is that the observed environment is stationary - which is not the case if multiple agents are present in the environment, each with their own policy that is being updated constantly. However, there are several options to circumvent this problem, for example creating independent 'oblivious' agents, having a centralized critic that evaluates the agents' actions, or simple communication structures between agents. Another aspect that has to be taken into account with MARL, is that the state space can grow rapidly when more agents are added.

In MARL, three different types of problems are defined: cooperative, competitive and mixed learning problems, where mixed problems are a combination of competitive and cooperative elements. The problem of this research project is a cooperative problem: the robots have no reason to compete with each other.

3.2.1 Independent agents

The simplest way of approaching the multi-agent learning problem is by training agents that are unaware of any other agents in the environment. This can be done, for example, by training a Deep Q Network (DQN) for each agent. This method ensures that the computational complexity increases significantly slower compared to when all agents observe each other's policies. However, there is also a big downside: the agents observe only their own (local) reward, and thus not actually learn to cooperate with each other. This is less of a problem in the competitive variant of MARL, but it is a significant issue in cooperative MARL.

3.2.2 Centralized critic

A solution to the problems of MARL begins with each agent learning their own policy, like with the independent agents. To make sure that the agents learn to work together properly, a critic can be added which evaluates each agent's policy, while having access to each agent's observations. This method has been quite successful in the past four years, and several different approaches are discussed in this section.

MADDPG

One of the first successful ways of applying a centralized critic was Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [41]. In a more recent implementation, MADDPG is used for UAV path planning [42]. This project, which is similar to this research, shows that good results can be obtained with MADDPG. However, the method becomes infeasible for a large number of agents - scaling up from three to five agents, the large computing times become prevalent. This is mainly caused by the fact that the critic evaluates the policies of all the agents simultaneously. MADDPG might be interesting for the cases with only a few agents, but not suitable for large robot swarms.

COMA

Counterfactual Multi-Agent (COMA) [43] uses a single centralized critic. It is a method that scales well due to the fact that its main complexity - and all of its learning - is in its critic network. The agents networks are merely used for inference. When choosing an action for an agent, a counterfactual baseline is used to single out that agent and keep all other agents' actions fixed. This is a solution to the MDP violations that MARL poses, and keeps the critic from evaluating all agent policies simultaneously, like in MADDPG. The method achieves good results, and is suitable for navigation-like cooperation tasks, as is shown in the practical applications.

Multiple actor-attention critic

In Multiple Actor-Attention Critic (MAAC) [44], each agent has their own critic. Because training each critic based on observations from all agents would be too expensive, an attention mechanism is created which directs the critics; they are given guidance to which agent they should pay attention for the learning process. The method is able to deal with larger number of agents than MADDPG successfully, although no computation times are reported.

Mean field reinforcement learning

Mean field reinforcement learning was first introduced by Yang et al. [45], and is based on mean field theory. The method was developed for dealing with large numbers of agents, and therefore naturally scales well. The main concept is as follows: as with other actor-critic approaches, there are two entities that determine an agents' policy: the agent itself and, in this case, the group dynamics of the agent's neighbours. In this way, a local group of agents functions as its own critic through their average actions, although the method could also be viewed as a communication method (see Section 3.2.3 below). Moreover, the authors also propose Mean Field Q-Learning (MFQ), which outperforms mean field actor-critic in multiple cases, despite its lower complexity. In practice, mean field RL has already been applied to solve a ride-sharing order dispatching problem, which has a few similarities to VRPs [46].

3.2.3 Learning to communicate

Another way of reducing the state space problem, is having a simple form of communication between agents, allowing them to exchange basic information on their policies. This communication is often learned behaviour itself, as shown for example by Foerster et al. [47] Learning to communicate can be used as an extension to other RL methods, although it can increase the computational complexity in a significant way, even when sharing small amounts of information. However, not much literature is available on using communication in path planning or navigation reinforcement learning problems.

Message dropout

Decreasing large input state spaces in reinforcement learning with communication can be done through applying message dropout [48]. In this method, a number of messages between agents are blocked, similar to how dropout can occur in real-world communication systems. This reduces the input space significantly, while not trading in performance. The authors test the method for communication-based MADDPG and various other communication approaches, but discuss that the method might be applied to non-communication-based methods such as COMA as well.

3.3 Other noteworthy approaches

In this section, a few other approaches to the path planning problem are discussed. The CPP problem is cast as a type of VRP in previous sections. However, there also exist other approaches to tackle CPP, of which one is described in this section. Moreover, a few machine learning approaches to solve path planning are covered that do not fit into the category of multi-agent reinforcement learning.

3.3.1 Coverage path planning using an SMT solver

Quite recently, the problem of multi-agent aerial monitoring was tackled in a different way [14]. In this project, a group of drones were deployed to monitor penguin colonies on Antarctica. Their coverage path planning approach called POPCORN tries to minimize backtracking, to make sure the drones' paths meet their battery constraints. A grid is constructed within a predetermined geofence, and this grid is modeled as a graph. A path is planned through the graph through an SMT solver, guided by various predefined constraints. This solution is fast, flexible and often produces better results than other CPP methods. Although this method is only not suitable for every scenario, it is still a highly promising one.

3.3.2 Reinforcement learning for path planning

Reinforcement learning can also be deployed directly for path planning, as is shown by the implementation by Zhang et. al. [49]. Their Geometric Reinforcement Learning (GRL) approach plans paths for single UAVs or groups of UAVs, taking into account threat areas (which could be obstacles or non-fly zones in agriculture) and collisions between agents. This risk calculation is combined with a straightforward grid-based path planning approach. Although not very well-documented, and only tested for up to three UAVs at the same time, this method can be useful for linking other approaches to the problem at hand.

Another reinforcement learning approach attempts to solve the VRP directly, through the use of a recurrent neural network decoder and an attention mechanism [50]. The main advantage of this approach is that only training the model is computationally expensive: if a model is successfully trained, new VRP scenarios can be solved almost instantly by the inference model. The results for this method are able to compete with those of other state-of-the-art VRP methods. Similarly, approaches to solve the TSP exist [51].

3.3.3 Deep learning for path planning

Apart from reinforcement learning, other machine learning methods can be used to approach path planning problems as well. For example, Mazzia et al. implement a deep learning approach called DeepWay for path planning in row-aligned crop management scenarios [52]. Although this method is only applicable to a specific scenario, in that scenario it is a light-weight and robust solution.

Recently, an approach that is applied in Natural Language Processing (NLP), a transformer network, was deployed successfully to solve TSP problems [53] in an efficient and near optimal way. Solving the TSP with deep learning is still a hot topic in combinatorial optimization, which is why probably more advances will be made in this field in the near future. Deep learning approaches have also been extended to multiple traveling salesmen, which is highly similar to the VRP [54].

Deep learning approaches to CPP also exist, such as the method by Yang and Luo [55]. Although this method is highly flexible, including obstacle avoidance, it scales badly to larger environments, which is why it is not considered any further. On the other hand, deep learning can also be used to control a robot's dynamics in path planning, often used for obstacle avoidance. This has been discussed as a possible approach as early as 1997 [56], where a neural network combined with a genetic algorithm steers a robot during path planning. However, this approach focuses mainly on the control part of the robots, which is outside the scope of this research.

3.4 State of the art summary

In this chapter, various state-of-the-art approaches to the path planning and target assignment problem have been considered. A comprehensive overview of the approaches and their relevant citations are shown in Table 3.1. Three main types of algorithms for solving the VRP have been discussed: exact, heuristic and meta-heuristic approaches. Coming in from a different angle, multi-agent reinforcement learning (MARL) and a few of its most promising methods are discussed. A few other related projects are discussed as well. Based on the literature found in this chapter, a number of methods are selected for testing, through various criteria. This is achieved through an algorithm comparison in the form of a design space exploration, shown in Table 4.1.

This research project contributes to the state of the art in the following ways:

- It focuses on complex agricultural topologies and considers a variety of real-world fields.
- It considers and compares various algorithms and methods, choosing the most suitable one for each type of field and number of agents available.
- It demonstrates an end-to-end solution, from the satellite image of the agriculture field to the target assignment of the agents.
- It evaluates the scalability of the methods used by testing for large numbers of landmarks and agents.

Category	**Approaches**
Exact solvers	Concorde [17] [18]
Heuristics	Nearest Neighbour (NN) [19], Christofides [21], k-opt [22] [23], Lin-Kernighan-Helsgaun (LKH) [24] [25], Clarke-Wright (CW) [26]
Metaheuristics	Simulated Annealing (SA) [28] [29], Tabu Search (TS) [29] [30], Guided Local Search (GLS) [31], Genetic Algorithm (GA) [33], Ant Colony Optimization (ACO) [35] [36], Particle Swarm Optimization (PSO) [37]
Multi-Agent Reinforcement Learning (MARL)	Independent Agents, MADDPG [41] [42], COMA [43], Multiple Actor-Attention Critic (MAAC) [44], Mean Field [45], Learn To Communicate [47], Message Dropout [48]
Reinforcement Learning and other Deep Learning	GRL [49], RL-VRP [50], RL-TSP [51], DeepWay [52], Transformer Network for TSP [53], Multi-Salesmen Pooling Network [54], DL-CPP [55], DL for Control [56]
Other	POPCORN [14]

Table 3.1: Overview of investigated state of the art methods and citations.

Chapter 4

Specification

In this chapter, the problem and approach are further specified. First, a design space exploration is done, which gives an overview of the discussed algorithms in Chapter 2 and results in a way to compare them against each other. Moreover, the requirements for the system design are given. Lastly, the assumptions under which the system operates are specified.

4.1 Algorithm comparison

To make a selection from the state-of-the-art algorithms that were investigated, a design space exploration is done in the following way. Each of the discussed algorithms (or groups of algorithms) is given a score in five different categories, which represent the relevance of the algorithm and the feasibility of including it into the comparison. This score reflects the expected performance of the algorithm in the respective category, based on literature and assumptions - note that this holds a significant amount of subjectivity. These scores will be used to make a primary selection of which algorithms to test.

The full design space exploration can be found in Table 4.1. The five categories on which the algorithms are scored are, from left to right: computational complexity, result quality, ease of implementation, scalability and consistency. *Computational complexity* and *scalability* reflect the amount of computing time necessary and how well the algorithm scales for larger problems. *Result quality* is an estimate of how good the average result of the algorithm will be (where ”good” means shorter path lengths and/or travel time). The score for *ease of implementation* depends on whether code for the algorithm is publicly available, and if not, how convoluted the method would be to implement manually. Lastly, *consistency* reflects how much deviation there will be between the algorithm’s results and/or performance, which could for example be caused by randomness or change with problem complexity. Each of these categories is also given a ***weight***, which is used to calculate a weighted average score - shown in the right-most column.

Algorithms are listed in order of appearance in Chapter 2. Each method includes the most relevant reference for a practical implementation. The given scores can either be - -, -, +/-, + or ++, represented by the integers -2 to 2 for calculating the total score and weighted average score.

4.1.1 Selected algorithms

To select the algorithms to be used for further testing, the average score of each algorithm is evaluated. A threshold is set; each algorithm that scores a + or higher (1 or higher) will be considered. That results in the following algorithms: Christofides and CW from the heuristic methods; all discussed meta-heuristic approaches except for PSO; and also POPCORN and RL-VRP. One exception are the MARL approaches; due to their complexity and how different they are from the other approaches, implementing them for the problem at hand is a difficult and time-consuming process. Therefore, only one is selected, the most promising one: Mean Field RL.

Christofides will be used as a benchmark: since it has a worst-case guarantee, it can be used as a baseline to compare the other test results to. POPCORN is only suited for coverage path planning applications without obstacles or predefined points of interest, but for those applications it is highly promising. The average score for message dropout is not above 1, however it might be useful to improve on MARL methods in a later stadium. The same holds for DeepWay, which is (like POPCORN) only suitable for a specific application, but promising for that application: row-based coverage path planning for ground robots.

Algorithm	Compl.	Res.Q.	E.o.I.	Scale	Cons.	Total score	**Avg. score**
Weight	***0.2***	***0.3***	***0.1***	***0.3***	***0.1***	***1***	
Exact methods*	- -	++	++	- -	++	2	**0**
Heuristic methods							
Nearest Neighbour [19]	+	+/-	++	+	- -	2	**0.5**
Christofides [21]	++	+/-	+	++	++	7	**1.3**
k-opt [22] [23]	+/-	++	++	- -	++	4	**0.4**
LKH [25]	+/-	++	+/-	+/-	+/-	2	**0.6**
CW [27]	+	+	+	+	+	5	**1**
Metaheuristics							
SA [32]	+	+	++	+	+	6	**1.1**
TS [32]	+	++	++	+	+	7	**1.4**
GLS [32]	+	++	++	+	+	7	**1.4**
GA [33]	+	+	+	+	+	5	**1**
ACO [36]	+	+	+	+	+	5	**1**
PSO [37]	+	+	+/-	+	+/-	3	**0.8**
Multi-agent RL							
Independent agents*	+	-	+	+	+/-	2	**0.3**
MADDPG [41]	-	++	-	-	+	0	**0.1**
COMA [43]	+	++	-	+	+	4	**1.1**
MAAC [44]	+	++	+/-	+	+	5	**1.2**
Mean field RL [45]	+	+	+/-	++	+	5	**1.2**
Mess. dropout** [48]	+	+	-	+	+/-	2	**0.7**
Other approaches							
POPCORN [14]	+/-	++	+	+	++	6	**1.2**
Geometric RL [49]	+	+	- -	+/-	+	1	**0.4**
RL-VRP [50]	+	++	+	+	+	6	**1.3**
DeepWay [52]	+	+	-	+	++	4	**0.9**

Table 4.1: Design space exploration. Scores given range from -2 (–) to 2 (++).
* Generalization or combination of various methods.
** Not really a method in itself, more of an extension.

4.2 Requirements

To properly answer the research questions, a number of requirements need to be formulated. This is done using a MoSCoW analysis [57], where each requirement is placed into one of the four MoSCoW categories: *Must* haves, *Should* haves, *Could* haves and *Won't* haves.

The system *must* have:

- A variable number of agents
- A representative cost function to evaluate problem solutions
- A link between real world and simulation; for example GPS coordinates as input and output

The system *should* have:

- Support for different types of topologies; for example obstacles, row-based crops, and random/clustered/grid-based points of interest
- Agent specifics and/or characteristics that are based on a real-world analogy
- At least one implemented method from each of the following categories, to observe the difference between different approaches: heuristic path planning approaches, meta-heuristic path planning approaches, multi-agent reinforcement learning, and other noteworthy approaches

The system *could* have:

- Mixed topologies; for example a combination of row-based crops and open field
- Asymmetric distance representations that model wind direction or slopes
- Partially obscured information, to represent a limited agent visibility range
- A Graphical User Interface (GUI) for accessible use

The system *won't* have:

- Agent dynamics and/or an accurate agent control system
- A dynamic planning environment

4.3 Assumptions

A number of assumptions need to be defined to create and evaluate the system. Some of these assumptions arise from the previous iteration of this project, others come from the aforementioned requirements. Possible future relaxations to these assumptions are discussed in Section 9.2.

- **There is only one type of agent.** All agents are either aerial robots or ground-based robots, a mix between the two is not considered.
- **The environment is static.** All environment conditions stay the same during simulation, except for the agents' location.
- **The environment is two-dimensional.** All relevant topology locations can be expressed in (x, y) coordinates. This means that aerial agents always operate from the same altitude.
- **There is only one agent origin, which is both a starting point and end location.** All agents start at the same location in the environment, which is also their end destination.
- **All environment information is globally available.** The environment layout is fully observable during simulation.
- **Agents always move at maximum speed.** This means that traveled distance by an agent is directly proportional to the time traveled. Stopping times at points of interest are not included, and wind / terrain friction forces are not taken into account.
- **The distance between two points is symmetric.** Slopes and wind direction are not taken into account; traveling from one point to another is exactly as fast as vice versa.
- **Accurate GPS coordinates are available and can be used to control the robots.** The system uses GPS locations as input and produces a list of target GPS coordinates as output for a robot to use.

Chapter 5

Experiment design

In this chapter, experiment specifics and the system pipeline are described. This includes the extraction of real-world topologies, the necessary calculations and further specification of the problem for some of the approaches used. Subsequently, the generated task scenarios for the tests are listed. Lastly, some details are given, for example the used agent specifics and the used hardware and software.

Figure 5.1 shows the full pipeline of the system, which takes a Google Earth topology as input, converts it to a simulation scenario, applies a number of methods and outputs the resulting solution in GPS coordinates and its cost. For more information about the cost function used, see Section 6.1. This schematic gives a global overview of the discussed points in this chapter. In Section 6.2.3, the addition of a GUI to the system is discussed, where the pipeline from Figure 5.1 is simplified into just a few inputs and outputs for the user.

5.1 Scenario setup

In this section, the setup for creating a test environment is described, including the following: importing a real-world topology, transforming the problem to a VRP and calculating the distance lookup table. Moreover, obstacle avoidance is discussed. This approach will be used to generate the scenarios and tasks described in Section 5.2.

5.1.1 Extracting topologies from Google Earth

Scenarios are based on real-world topologies extracted using Google Earth. How to construct such a topology is described in Appendix B. The resulting `.kml` file is loaded into the simulation environment, where the GPS coordinates are scaled such that the topology fits into a coordinate system ranging from -1 to 1 on both axes and

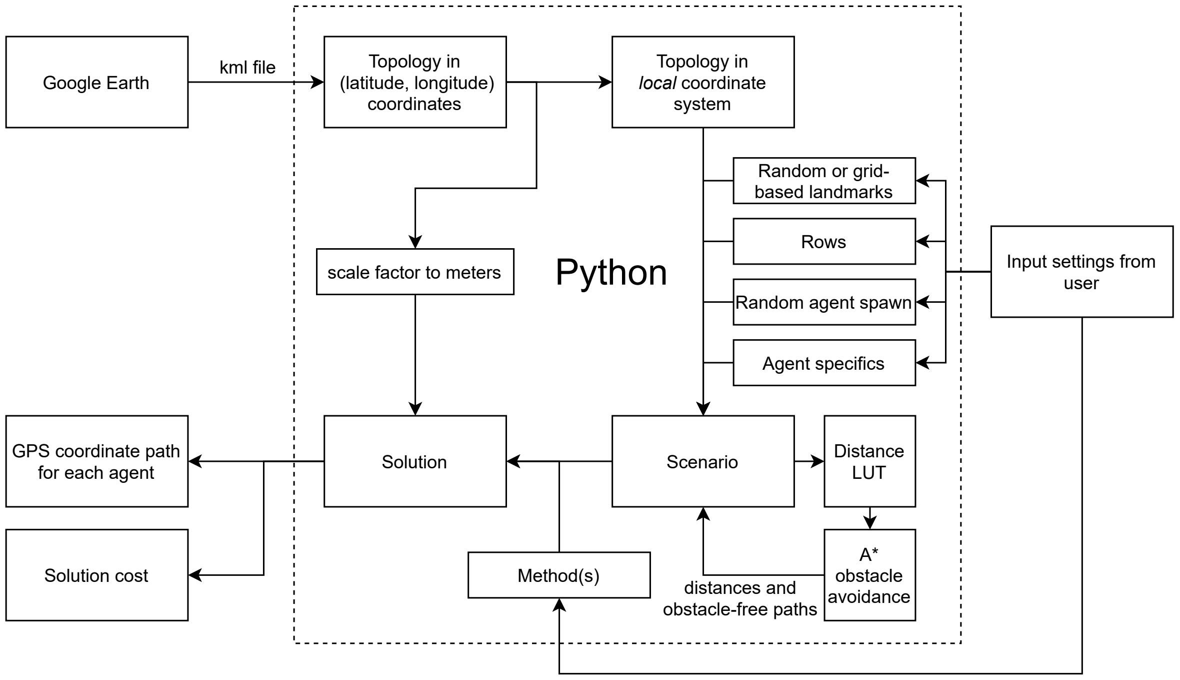


Figure 5.1: System pipeline overview

the origin at the center ($[-1,1] \times [-1,1]$) - which will from hereon be referred to as the *local* coordinate system. During this process, a scale factor to convert distances in the local system to meters is stored. This scale factor is calculated using the haversine formula[1]:

$$a = \sin^2(\frac{1}{2}\Delta\phi) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2(\frac{1}{2}\Delta\lambda)$$

$$d_{1,2} = 2R \cdot \text{arctan2}(\sqrt{a}, \sqrt{1-a})$$

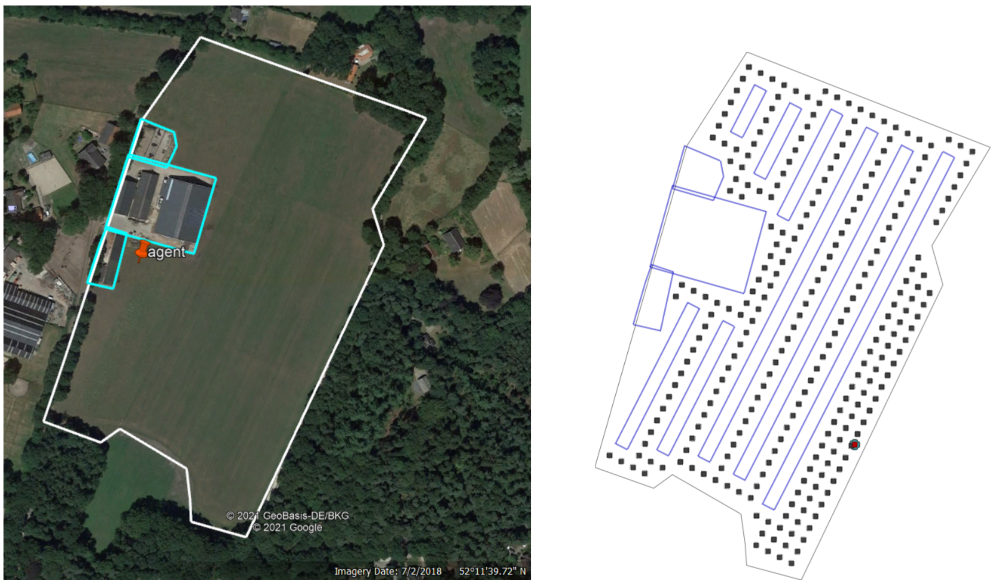
With R the mean radius of the earth in meters, ϕ the latitude and λ the longitude, both in radians. The result $d_{1,2}$ is the distance between two lat-lon coordinate pairs in meters.

The distance in meters is calculated both in the horizontal and vertical direction, and these are combined into the scaling factor. This scale factor results in an approximate conversion factor from local distances to meters, the error margin of which is discussed futher in Chapter 8. The normalization factors used in the coordinate system conversion can be reverted at the end to transform a path planning solution back to GPS coordinates.

After coordinate normalization, landmarks are added either randomly or based on a grid. Row-like obstacles can be added as well (based on the same grid) to simulate row-based crop fields. Moreover, the agent origin (or *spawn*) can be randomly

[1] `http://www.movable-type.co.uk/scripts/latlong.html`

relocated. The spawn and landmarks are always initialized such that they lie within the field boundary and not inside any obstacles. A topology in Google Earth and the resulting scenario in the simulation environment can be seen in Figure 5.2.



(a) topology `b.kml` *(b)* scenario `bgrid40`

Figure 5.2: Google Earth topology and resulting simulation environment. The landmarks (grey dots) are based on a grid. The agent origin (red dot) is randomly relocated. Row obstacles are added to simulate rows of crops.

5.1.2 Precalculated distances and obstacle avoidance

Most of the methods that are used for testing make use of a precalculated distance Lookup Table (LUT), which is an n by n matrix which stores each distance between two points. The LUT is calculated during the setup of the environment, where the Euclidean distance is used to find the straight line distance between two points:

$$d_{i,j} = \sqrt{(j_x - i_x)^2 + (j_y - i_y)^2}$$

However, direct paths between points might be infeasible, for example when they pass through an obstacle or cross the topology boundary. If this is the case, an A* search [58] [59] is deployed to find a valid path between the two points. For this, an underlying grid is used, of which the nodes are used for path planning in 8 directions (horizontally, vertically and diagonally). The resulting path is stored in another lookup table, which is used for rendering and producing output coordinates. The total length of the found path is stored in the aforementioned distance LUT. Both the direct paths and the obstacle avoidance paths distances are stored in both directions, since the problem is symmetric.

In rare cases, A* search can fail to find a route, for example if a point is unreachable via the underlying grid due to obstacles or the world boundary. In that case, the edge (path between points) is considered invalid and a large value[2] is added to the distance LUT to represent the infeasible cost of taking that edge. Points that have invalid edges to more than 25% of the other landmarks are considered infeasible and discarded, or in the case of random initialization, randomly relocated until the landmark is considered feasible. This *point selection* is based on preliminary results obtained for different methods, since some path planning methods try to take invalid edges if there are too few valid options to reach a point, regardless of whether enough valid edges are available. With 25% of the edges being valid, there is enough leeway for the methods to never take an invalid edge wrongly, but it does not oversimplify the environment. Note that A* search not finding a path happens rarely. As an example of the procedure, in scenario `bgrid40` (Figure 5.2b) there were a few points in the bottom left corner that had to be removed according to this heuristic since they could only find valid paths to each other and not to the rest of the points. The entire point initialization and selection procedure is summarized in Figure 5.3.

All these calculations can add up in computation time, which will be taken into account for the methods that use the functionalities. To avoid doing the calculations each time, the tables and obstacle avoidance paths can be stored locally so that they can be loaded from file almost instantly when necessary.

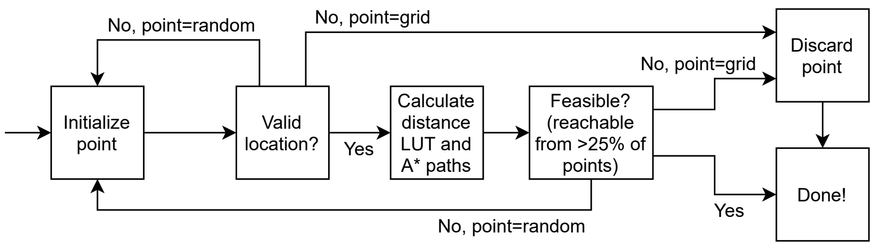


Figure 5.3: Point generation and selection procedure.

5.1.3 VRP specification

Most of the algorithms to be tested require a specification of the problem as a basic Vehicle Routing Problem. The agent origin is considered the depot, and the landmarks (points of interest) are the customers. The distance LUT from the previous section is used, and a fixed number of vehicles (agents) is available. Note that some methods (CW, POPCORN, RL-VRP and all the OR-Tools methods) determine how

[2]1000 times the maximum distance possible in the local coordinate system ($2000\sqrt{2}$)

many vehicles to use independently. If less vehicles are needed than there are available vehicles, the excess agents will stay at the depot. If more vehicles are needed than available, the shortest two routes are combined iteratively until the number of used vehicles matches the number of available vehicles.

Several of the used VRP solvers solve the CVRP, which means each customer is assigned a demand and each vehicle is assigned a maximum loading capacity. This functionality could be deployed in the future to accommodate, However, for these methods the demand of each customer is set to 0, such that each vehicle is allowed to visit an infinite number of customers. The vehicles are limited by another factor: their maximum allowed travel distance, which is shown in Table 5.2. There are two methods (GA and RL-VRP) which do not use a demand of 0, but instead an equal distribution of demand across all customers in order for them to function properly. See Section 6.3 for more details.

5.2 Scenarios

In order to find the best method for different agricultural tasks, five tasks will be simulated, based on the available options for creating a scenario. Coverage path planning (CPP) tasks have the landmarks arranged in a grid, to structurally cover as much of the field as possible. Precision tasks use instead randomly scattered landmarks, either fully random or in clusters. The ground-based tasks have row obstacles added in to simulate row-based crops.

For each of these tasks, two scenarios are constructed: one large problem (around 200 landmarks) and one small problem (around 50 landmarks). In Table 5.1, more specifics of these scenarios are presented. In Figures 5.4-5.8 the layout of each scenario can be found. Each scenario is based on a real-world agricultural field, manually extracted from Google Earth as described in Section 5.1.1.

Task	**Large**	**Size**	**Small**	**Size**	**Agents**	**Other**
Aerial CPP	`bgrid20`	126	`cgrid10`	54	Aerial	-
Ground-based CPP	`bgrid40`	262	`dgrid20`	116	Ground	Rows
Aerial precision	`z200`	200	`e50`	50	Aerial	-
Aerial precision (cluster)	`hcl200`	200	`acl50`	50	Aerial	Clustered
Ground-based precision	`i200`	200	`l50`	50	Ground	Rows

Table 5.1: Overview of the different tasks and problem sizes.

5.3 Robot specifications

A number of constants are used to describe the specifications of the agents, taking into account the specified assumptions from 4.3. The aerial agents are modeled after the Parrot ANAFI[3]. Ground robots are modeled after the AVO weeding robot by Ecorobotics[4]. The used variables can be found in Table 5.2. Given the assumptions, the maximum distance is simply calculated by multiplying the battery life and the maximum horizontal speed. The vertical speed and range of the robots are currently not used, but have been included for completeness. In Section 9.2, a use for these variables is discussed.

Agent model	Battery	Hor. speed	Ver. speed	Max. distance	Range
Aerial (ANAFI)	1500 s	15 m/s	4 m/s	22.5 km	4000 m
Ground (AVO)	8 h	1 m/s	-	28.8 km	∞

Table 5.2: Specifications used for modelling the agents

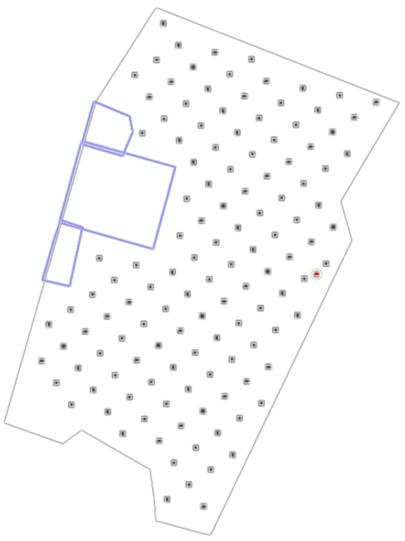
5.4 Hardware and software

The experiments are performed on the computing cluster of the CTIT group at the University of Twente, which has various types of GPUs and CPU cores available. Due to how the experiment tasks are scheduled, it is not possible to determine which of these hardware components are used for which experiment.

All experiments are done using Python 3.8. GPU support is added using NVIDIA CUDA 10.1[5]. Google Earth Pro 7.3.3[6] is used to set up the scenarios.

For a full list of used Python packages and external repositories, see Appendix A.

[3]`https://www.parrot.com/en/drones/anafi/technical-specifications`
[4]`https://www.ecorobotix.com/en/avo-autonomous-robot-weeder/`
[5]Available from `https://developer.nvidia.com/cuda-10.1-download-archive-base`
[6]Available from `https://www.google.com/earth/versions`


(a) scenario `bgrid20`

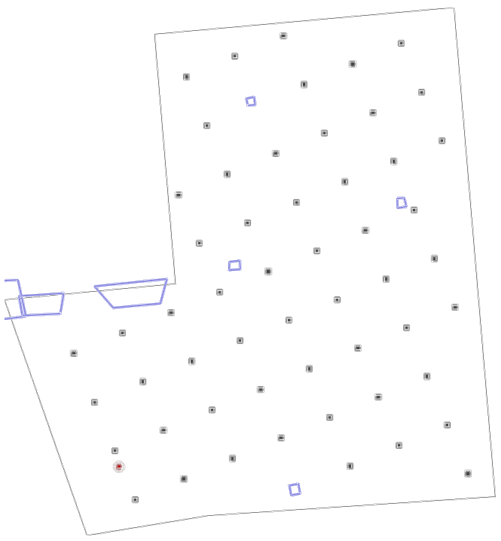
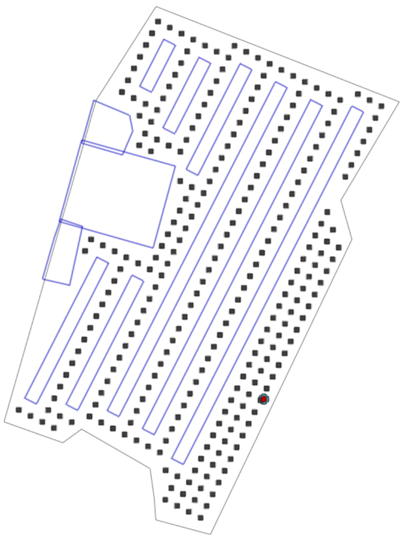

(b) scenario `cgrid10`

Figure 5.4: Aerial coverage path planning tasks.


(a) scenario `bgrid40`

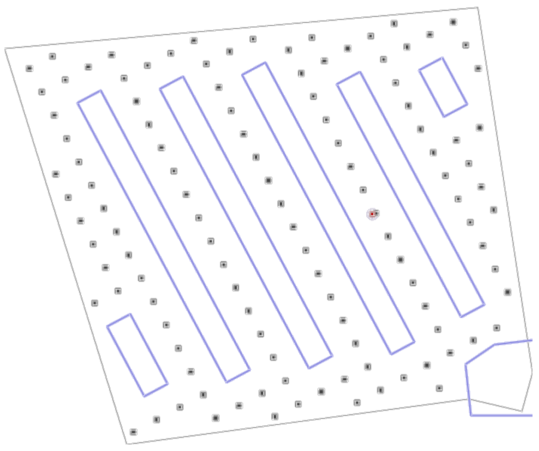
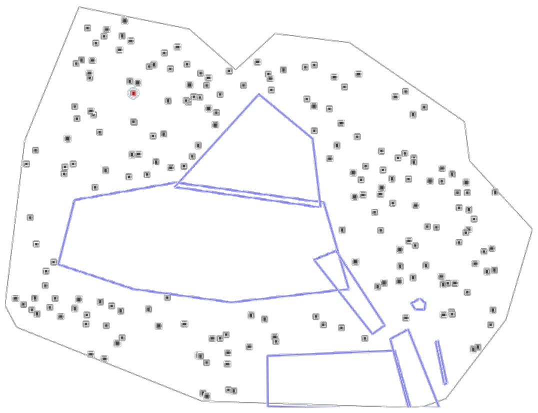

(b) scenario `dgrid20`

Figure 5.5: Ground-based coverage path planning tasks with rows of crops.


(a) scenario `z200`

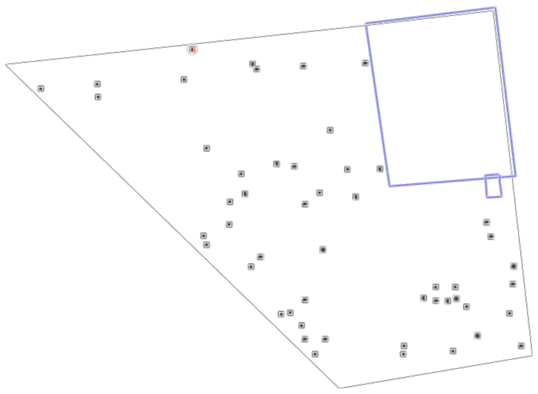
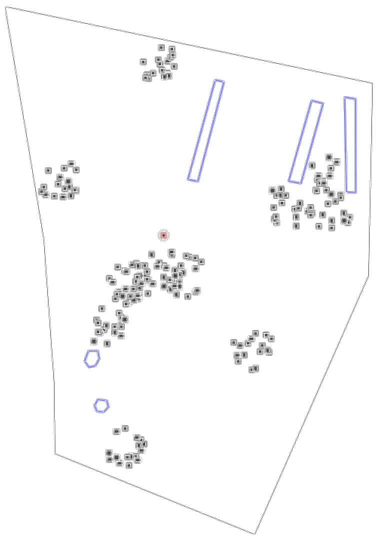

(b) scenario `e50`

Figure 5.6: Aerial precision tasks.


(a) scenario `hcl200`

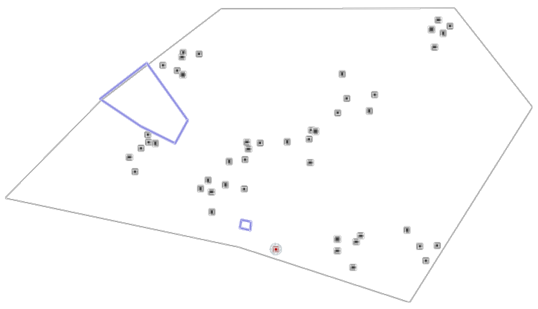
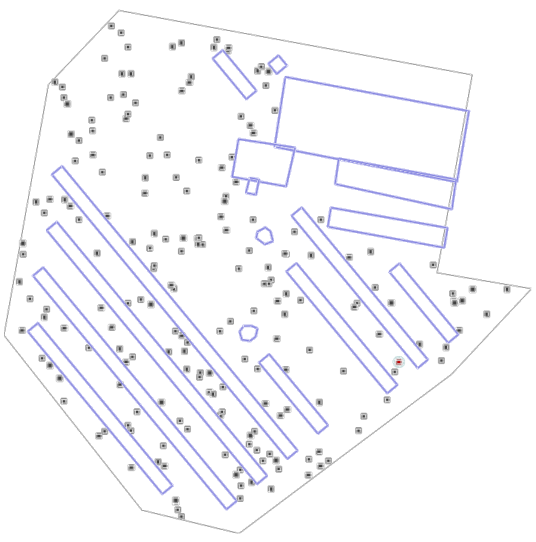

(b) scenario `acl50`

Figure 5.7: Aerial clustered precision tasks.


(a) scenario `i200`

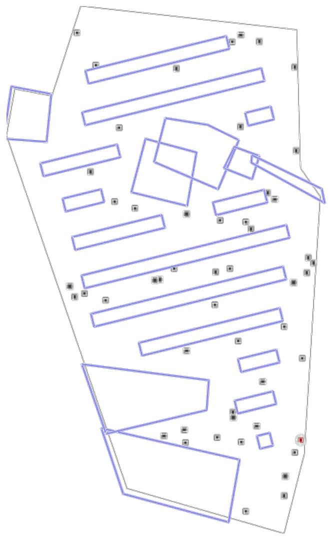

(b) scenario `150`

Figure 5.8: Ground-based precision tasks with rows of crops.

Chapter 6

Method

In this chapter, the methodology for answering the research questions is given. First, a definition of the cost evaluation function for the generated solutions is given. Next, the testing approach is explained, as well as how the research questions will be answered. Lastly, details and parameter settings are provided for each evaluated algorithm.

6.1 Cost evaluation function

To objectively evaluate the performance quality of an algorithm, a cost function has to be defined. A function is chosen which represents not only the distance or time traveled by the agents, but also the fairness or standard deviation between agents. This is done to evaluate whether the paths are divided evenly between agents.

Given are m, the total number of agents, and D, the set of distances travelled by each agent. Then, the average distance per agent d_μ and the standard deviation in distance between agents d_σ can be calculated as follows, both in meters:

$$d_\mu = \frac{\sum D}{m} \qquad d_\sigma = \sqrt{\frac{\sum (D - d_\mu)^2}{m}}$$

The average and standard deviation ignore empty routes, i.e. any route k is not considered if $d_k = 0$.

The maximum of D is the distance traveled by the slowest agent, which is directly proportional to the total operation time because of the assumption that the agent always travels at maximum speed. The cost function will be as follows, balancing total operation time against a combination of the average agent distance and standard deviation between agent distances:

$$cost = z \max D + (1 - z)(q d_\mu + (1 - q) d_\sigma)$$

With balancing factors q and z, which are constants:

$$0 \leq q \leq 1 \qquad 0 \leq z \leq 1$$

This cost function is partially based on the cost function in the previous iteration of the project, which comes from the work of Seyyedhasani and Dvorak [15]. For the experiments, the following values for q and z are used, to represent an equal balance between solution speed and fairness between agents:

$$q = z = 0.5$$

The value for q and z can be used to shift the balance of the cost function towards the more important factors for the target application, for example based on what type of agents are used. However, the more general equal balance is used for the experiments in this research.

6.2 Test goals

In this section, the approach for answering the research questions is given. Each of the sections corresponds to one of the subquestions defined in Section 1.4. The three of these methods combined will result in a design which answers the main research question.

6.2.1 Evaluating method scalability

To evaluate how each of the methods selected in the design space exploration scales, computation times are recorded for each of them. Two parameter sweeps are done. The first is for an increasing problem size, where the number of agents is kept constant (10) and the number of landmarks is increased from 10 to 270 in increments of 10. The second sweep uses a fixed number of landmarks (100) and increases the number of agents from 1 to 50 in increments of 1. All this is summarized in the top part of Table 6.1. Both of these parameter sweeps use the same base topology (`h.kml`, see also Figure 5.7a), where the agents are aerial and the landmarks are distributed randomly and not clustered. This topology is chosen because it is relatively straightforward: the obstacles are not taken into account because the agents are aerial, and the boundary has a relatively simple shape. However, any of the topologies discussed in this chapter would have been suitable for these measurements - except when testing RL-VRP, since it does not get obstacle information as explained in 6.3.

Parameter	Smallest	Largest	Increment
Agents (aerial, 100 landmarks)	1	51	1
Landmarks (10 agents)	10	270	10
Optimal num. agents per topology	2	30	2

Table 6.1: Parameter sweep settings

6.2.2 Finding the best method for each topology

To find the best method for each topology, first the optimal number of agents for the scenario is determined. This is either done as a part of the method itself (for CW, RL-VRP and POPCORN), or by doing a parameter sweep for the number of agents and evaluating the results using the cost function explained in 6.1. The sweep values are given in the bottom part of Table 6.1.

Next, two main variables are measured and recorded for the final comparison: the previously mentioned cost and the computation time. Based on the resulting values for these variables, conclusions can be drawn on the suitability of each method for the specific topology. Apart from this, to observe the relative performance of each algorithm, the total agent operation time for each solution is calculated as follows, with v the agents' horizontal velocity:

$$t_{total} = \frac{\max D}{v}$$

This t_{total} will be evaluated for each method by comparing it to the average operation time for each topology. This is done by calculating the percentage difference from the average t_{total}.

6.2.3 Towards a real-world application

To ensure that the simulations done represent realistic scenarios, the topologies are based on real-world agricultural fields. Moreover, the agent specifics are based on real robots that could be deployed in such scenarios. To ensure that the step towards real-world implementation is made, the possibility is added to convert path planning solutions to GPS coordinates, which can then be used for robot navigation given that the robot has access to accurate GPS data.

Finally, a simple Graphical User Interface GUI is made that increases accessibility for two tasks: both for using the system to solve new path planning problems using the tested algorithms, and for repeating the experiments done in this project. With a GUI added, the system pipeline as shown in Figure 5.1 can be changed to match the one in Figure 6.1.

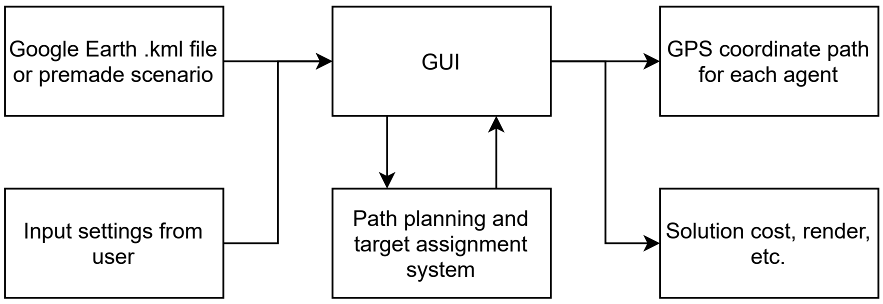

Google Earth .kml file or premade scenario
GUI
GPS coordinate path for each agent
Input settings from user
Path planning and target assignment system
Solution cost, render, etc.

Figure 6.1: Simplified system pipeline with added GUI.

6.3 Algorithm details and parameters

In this section, the specifics for each approach used during the experiments are explained in more detail. For each method, the relevant section in Chapter 3 is included as well.

Christofides

See Section 3.1.2. Christofides' algorithm [21] is in itself a method for finding single TSP path. To apply it to a VRP, the points have to be subdivided between the available vehicles before calculating the TSP solution for each vehicle separately. This is done using k-means clustering [60], where k is set equal to the number of available agents. After clustering, the path through all points in the cluster is connected to the agent spawn by finding the point closest to the spawn, and making that the starting and return point of the route.

Clarke-Wright

See Section 3.1.2. The Clarke-Wright savings heuristic [26] is deployed with the parallel approach (as explained in [27]). As mentioned previously, the demand for all points is set to 0 such that each vehicle can visit an infinite number of points until its distance limit (battery life) has been reached.

Google OR-Tools

See Section 3.1.3. The routing suite of Google OR-Tools [32] contains three of the meta-heuristics to be tested: Simulated Annealing, Tabu Search and Guided Local Search. The same settings are used for all three methods. The initial solution is generated using the `PATH_CHEAPEST_ARC` setting, which is a solution based on the Nearest Neighbour heuristic. A time limit is used as a stopping criterion, and it is determined through the following equation:

$$t_{max} = \frac{n^3}{2^8} \text{ if } n > 50, \text{ else } t_{max} = \frac{50^3}{2^8}$$

In this way, the problem size n determines the time limit (in seconds). Problems for which $n \leq 50$ are all allowed to use the time limit for $n = 50$, of around 490 seconds. This time limit formula is based on two factors: the time needed to converge to a good and possibly near-optimal solution, and the time available for running the experiments. It is definitely possible that, with larger computation times, better solutions are found.

Genetic Algorithm

See Section 3.1.3. The Genetic Algorithm VRP approach of Baker and Ayechew [33] is used, with a few modifications. The initial solution uses an equal demand distribution between points to ensure all vehicles are used. The demand for each point is chosen such that each vehicle is allowed to visit n_v points, where:

$$n_v = \frac{n}{m}$$

with n the number of points and m the number of vehicles. Moreover, the cost function from Section 6.1 is used instead of the fitness function used in the paper. As in the paper, small problem sizes ($n \leq 50$) use a population size of 30 and larger problems use a population size of 50.

The algorithm stops reproducing solutions either when no improvement is found in the last 100,000 generations, or when the computation time limit has been reached. The time limit is calculated in a similar way as for the OR-Tools approach, but with a different formula, since the GA often converges faster than OR-Tools (based on some initial observations).

$$t_{max} = \frac{n^3}{2^{10}} \text{ if } n > 100, \text{ else } t_{max} = \frac{100^3}{2^{10}}$$

Again, t_{max} is given in seconds. This formula results always yields a t_{max} that is 4 times smaller than the t_{max} of Google OR-Tools, however, problems for which $n \leq 100$ are all allowed to use the computation time of $n = 100$, which is around 980 seconds.

Ant Colony Optimization

See Section 3.1.3. The version of ACO used is based on the one used in the previous version of this project [9] [61] [62]. Although parameters should be adjusted to match each specific scenario, the same parameters are used for all experiments.

These are based on the optimal parameters found by Gaertner and Clark [63] for a specific TSP instance of comparable size to the smaller problems, and are shown in Table 6.2. Moreover, the ants are allowed to share pheromone data. Although these parameters might be far from optimal for the experiment scenarios, they present a general starting point with which to test the ACO performance.

The way the scenarios are set up, it is not possible to produce results for ACO for the clustered tasks (`acl50` and `hcl200`). Because the points are grouped far away from each other, ACO does not converge to a solution for these scenarios.

Parameter	α	β	ρ	q	tolerance	max. iterations
Value	3	12	0.6	0.2	$5 * 10^{-2}$	50

Table 6.2: Parameters used for Ant Colony Optimization

Mean Field Reinforcement Learning

See Section 3.2.2. An attempt is made to apply mean field reinforcement learning to find path planning solutions. The Mean Field Q-Learning approach (MFQ) is used, with the DQN network that the original authors use to solve the battle game [45]. The network is simplified to contain just two dense layers, of 256 units each. One network is instantiated for each agent.

Each agent has their own observation and reward. For the observation, the agent is allowed to observe only a limited number of landmarks. Only the n closest *unvisited* landmarks are observed, where $n = 5$ for the initial experiments. If not enough unvisited points are available, the observation is appended with the agent starting location to keep it the same size. The action each agent can take corresponds with visiting the landmarks from the observation. Thus, if 5 landmarks are observed, the agent can take 5 different actions: move to either of them instantly - there are no dynamics used.

The agent observation consists of three pieces of information per possible action: the distance to the point (taking into account obstacle avoidance), whether there is another agent present at that point, and the mean field action from the other agents - where only the 5 closest agents are considered. The reward the agent receives are as follows: it gets 1 point whenever it visits an unvisited location, and -0.01 points for every move multiplied by the distance of that move. The agent gets -0.1 points whenever it stays in the same location, which only happens at the end when not enough unvisited locations are left. Each episode is done once all points have been visited, or if 1000 timesteps have passed (1000 actions taken by each agent).

With all of the above settings, several training setups are tested. However, none of these seem to converge to a good cost result. As an example, one of the training curves is shown in Figure 6.2. Because of time constraints for the experiments, MFQ will not be compared to the other results in Chapter 7, since no viable results can be produced for this method yet. Options for how to get the training to converge are discussed in Section 8.2.

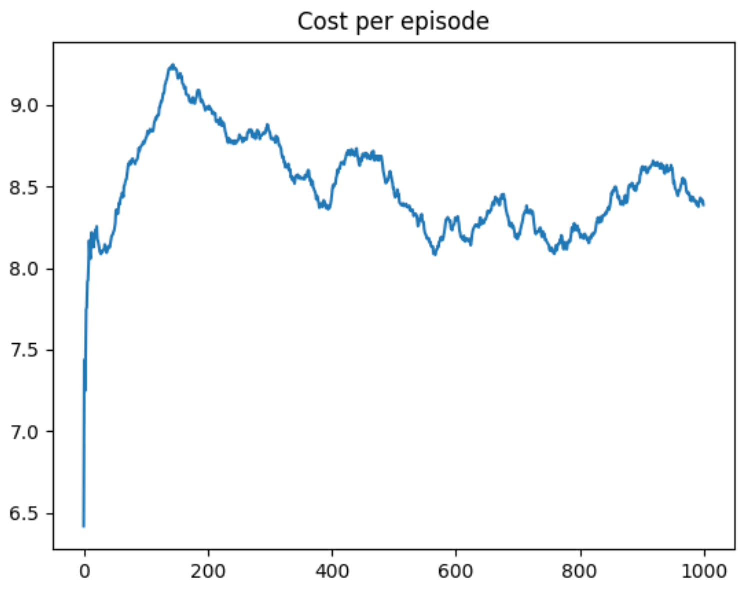


Figure 6.2: Result for training five MFQ agents on random problems with $n = 50$. The running average of the cost is plotted on the y-axis, and the number of episodes is on the x-axis.

RL-VRP

See Section 3.3.2. For the RL-VRP approach [50], the greedy approach is used. A model is trained for various problem sizes, with demand constraints based on the values supplied in the paper. An overview of the trained models can be found in Table 6.3. How well each model performs on the test set is shown in Figure 6.3, where the average path length is the average over the full test set (1000 instances).

The model closest to the problem is selected, with regards to the number of nodes. This model is used for finding an inference solution. Like with the GA approach, the demand for each point is chosen such that each vehicle is allowed to visit n_v points, where:

$$n_v = \frac{n}{m}$$

with n the number of points and m the number of vehicles. Moreover, the coordinates of all locations are transformed such that they all fit in the $[0, 1] \times [0, 1]$ unit square, to match the network input used in the paper. Note that RL-VRP takes as input only the location of each node and its demand, and thus the precomputed paths and distance LUT are not used for this approach.

Number of nodes	10	20	30	40	50	70	100	150	200	250
demand	20	30	34	37	40	45	50	60	65	70

Table 6.3: Trained models for the RL-VRP approach.

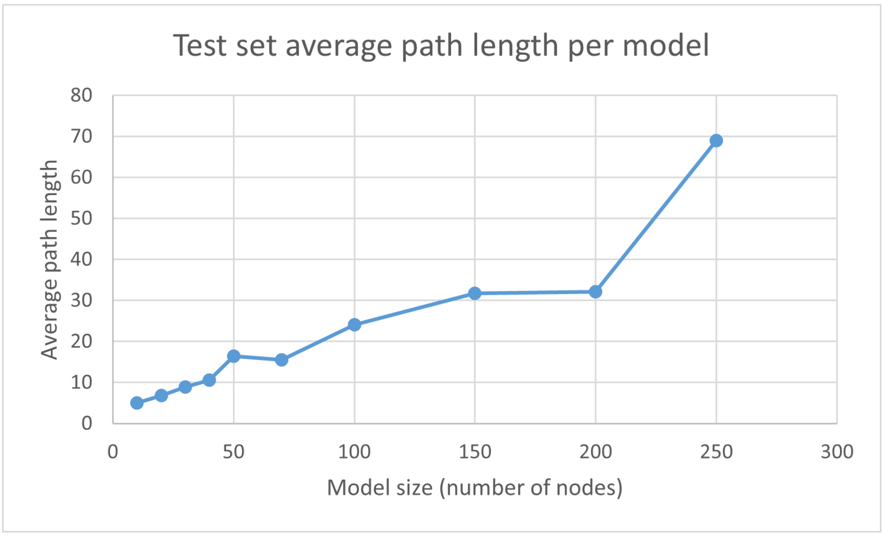


Figure 6.3: Average test set results for each of the trained RL-VRP models.

POPCORN

See Section 3.3.1. The POPCORN approach [14], or `wadl-planner` is deployed directly from its respective python package. The system takes the agent specifics and GPS coordinates of the field boundary as inputs, and the grid size is set equal to the used grid size in the scenario. The method then produces its own grid for doing coverage path planning, which has approximately the same number of landmarks, making the solution comparable to a solution using the actual points in the predefined scenario task. As mentioned before, POPCORN will only be deployed to solve the aerial CPP tasks, namely `bgrid20` and `cgrid10` (Figure 5.4).

Chapter 7

Results

In this chapter, the results are shown after applying the methodology from the previous chapter. First, the scalability of each algorithm is reported with regards to the number of landmarks and number of agents. Next, computation times and scores are reported for each of the predefined agricultural tasks. Lastly, the real-world application of the system is explained.

7.1 Method scalability

In this section, the effect of increasing the number of landmarks and the number of agents is discussed, in an attempt to visualize how each method scales. Note that the POPCORN method is not included at all in this chapter. This is because it fits only a specific type of scenario and it is hard to control the number of landmarks used in one of its CPP surveys. Also note that for the time limited approaches, OR-Tools and GA, the full parameter sweep is not completed. This is due to the limited amount of experiment time available.

7.1.1 Scaling of computation time

In Figure 7.1, the computation time is plotted against the number of agents. As expected, GA stays at its fixed time limit for 100 landmarks, at around 1000 seconds. ACO scales badly against the number of agents, quickly increasing in computation time. Christofides performs the best, with almost no time necessary to produce the solution. Note that POPCORN, RL-VRP and Clarke-Wright and the OR-Tools approaches are excluded. Because they optimize their own number of agents, it is hard to reproduce this sweep experiment for them in a meaningful way.

In Figure 7.2, a similar graph is shown, but now with an increasing number of landmarks. Christofides shows that it scales very well, together with Clarke-Wright.

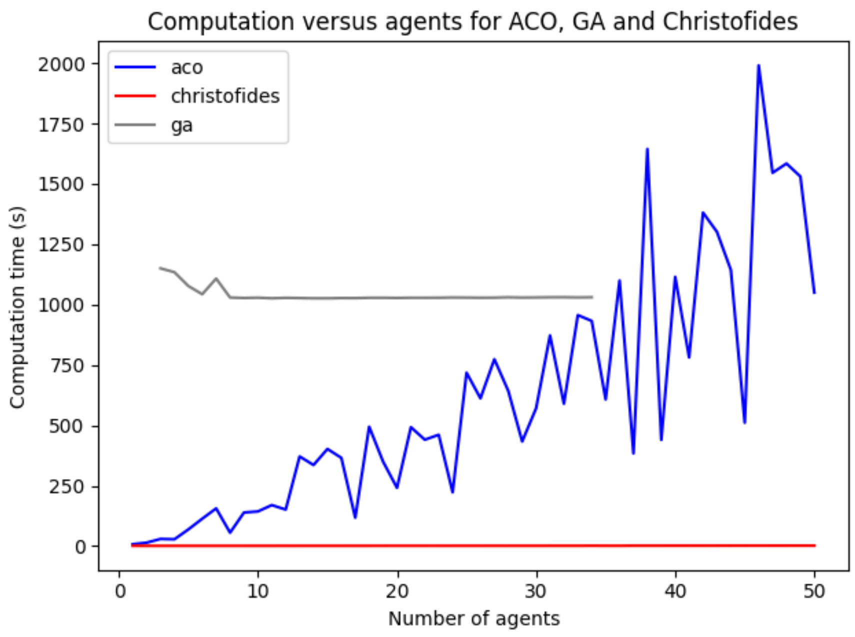


Figure 7.1: Computation time measured against an increasing number of agents, for a topology with 100 landmarks.

RL-VRP scales the best, which is expected when only using inference on a pre-trained network. In contrast to how it scales to the number of agents, ACO scales fine with regards to the number of landmarks.

To be able to see how the slower metaheuristic algorithms differ from these results, they are plotted separately, in Figure 7.3. This plots includes all methods which use a time limit. As GLS, SA and TS have the same formula for determining their time limit, they yield almost exactly the same result. GA uses a different formula, which can clearly be seen in this plot.

7.1.2 Scaling of cost

How well the cost scales against the number of agents is shown in Figure 7.4. Since adding extra agents is not penalized directly in the cost function, it can be seen that after adding a certain number of agents for each method, the cost levels out at a certain level. For more stochastic approaches such as ACO and GA, this level is more noisy than for Christofides. As in Figure 7.1, the Google OR-Tools methods, RL-VRP and POPCORN are excluded since they optimize the number of agents used by themselves.

As a final evaluation of how each method scales, the number of landmarks is compared to the cost (Figure 7.5). As can be seen, none of the methods' costs scale badly against the number of landmarks. Given enough computation time, each

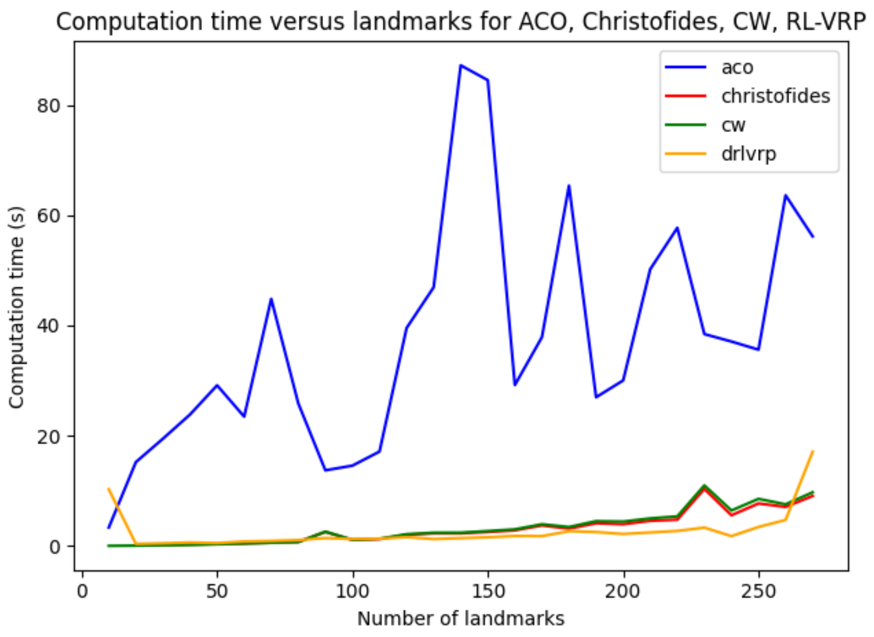


Figure 7.2: Computation time measured against an increasing number of landmarks, with 10 agents available for each method.

method should be able to produce similar results between different problem sizes. However, there is a clear difference in performance level between the algorithms. ACO performs the worst, followed by RL-VRP. The OR-Tools methods get the best scores for most topologies, with some exceptions where Christofides and/or CW score better.

7.2 Best method per topology

Next, results for the method comparison per predefined environment are given. First, for each method the optimal number of agents is determined. Next, a solution is generated for each environment, with that number of agents.

7.2.1 Optimal number of agents

As described previously, the optimal number of agents is determined by doing another sweep for the number of agents, this time per environment. This approach is only used for the methods that do not optimize the amount as a part of their approach. The optimal amount of agents is found by inspecting plots such as the one in Figure 7.6. If it is unclear which value is the best, the value with the smallest number of agents is chosen - using more agents than necessary is never considered positive. A collection of all plots used can be found in Appendix C.

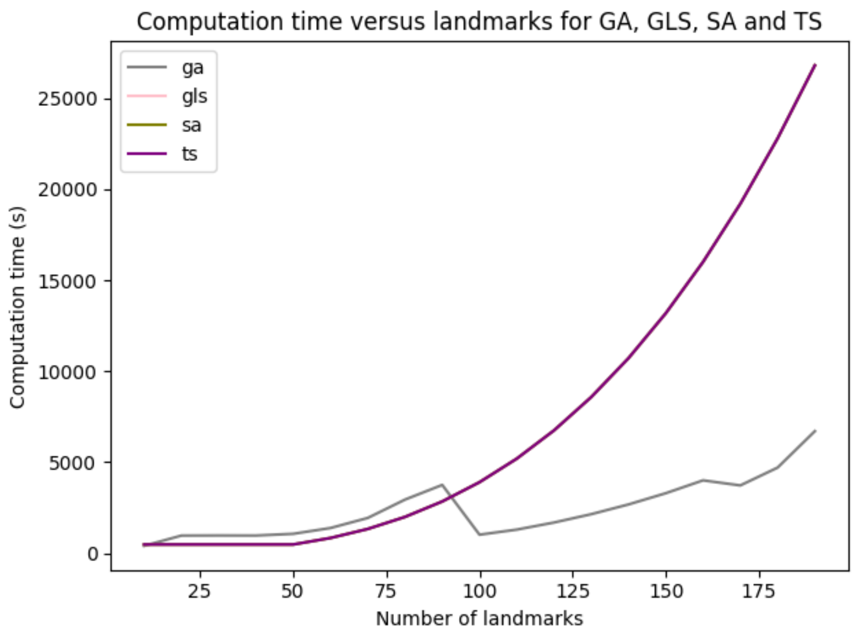


Figure 7.3: Computation time measured for GA and OR-Tools against an increasing number of landmarks, with 10 agents available for each method.

An overview of the optimal number of agents for each method per scenario can be found in Table 7.1. For POPCORN (SMT), RL-VRP (DRL), Clarke-Wright (CW), Simulated Annealing (SA), Tabu Search (TS) and Guided Local Search (GLS), the number of agents is determined by the method itself. Note that the amount of agents for `bgrid20` and `hcl200` when applying RL-VRP are infeasible - this might be because the models for larger sizes have not been optimized fully. Moreover, Christofides can obtain solid results (the best even in some cases, as will be discussed further on), but it needs a large number of agents to do so compared to the other methods. The large number of agents needed also holds for the other methods that do not optimize the number of agents internally, namely the Genetic Algorithm (GA) and Ant Colony Optimization (ACO).

7.2.2 Best method

Now that the optimal number of agents has been determined, the final tests for each scenario can be performed. The results of these tests are shown in Table 7.3 and Table 7.2. Moreover, the difference from the best operation time per topology is shown in Table 7.4, where 0% represents the best total operation time. For completeness, a scatter plot is made for each topology, plotting the cost against the computation time for each scenario and per method, like the one in Figure 7.7. The other scatter plots can be found in Appendix D.

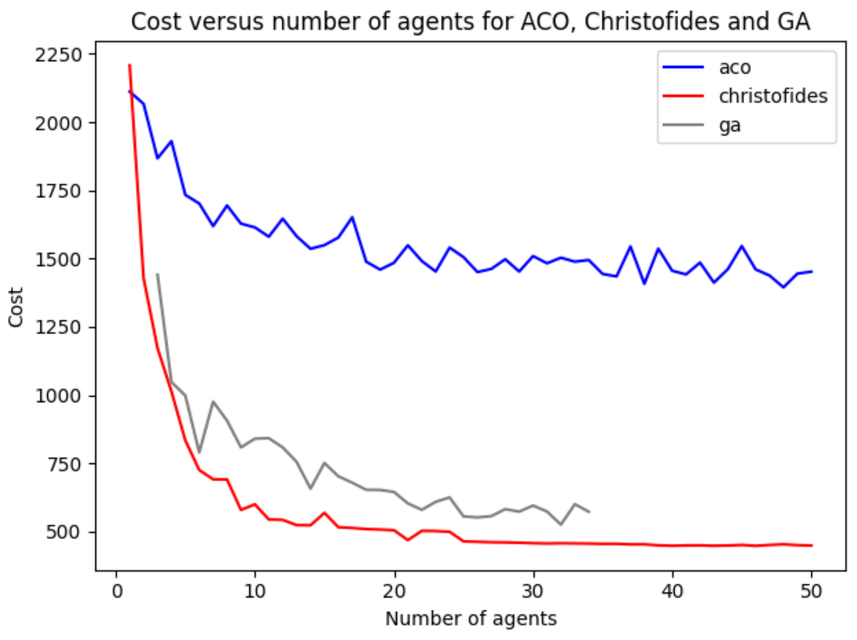


Figure 7.4: Cost measured against an increasing number of agents, for a topology with 100 landmarks.

Note that for RL-VRP (*DRL* in the tables) the results for ground-based instances are not entirely representative; the network did not know where the obstacles were, but in the calculating the cost, the obstacle avoidance paths of the other methods were used.

As expected based on the results from the parameter sweep in 7.1, RL-VRP is by far the fastest method for most tasks, mainly because inference is fast but also due to the fact that the precalculation times do not have to be taken into account since no distance table or obstacle avoidance is used. This can be observed in Table 7.2. Christofides and Clarke-Wright are also fast, but have to take precalculation times into account for using the distance tables.

Christofides appears to be the best method cost-wise for some of the topologies, but it uses a relatively large number of agents to get to these numbers. Clarke-Wright also obtains good scores for most topologies, while keeping the number of agents low. The best scoring method overall is Tabu Search, with Guided Local Search a close second. However, Clarke-Wright, Christofides and Simulated Annealing can compete with this, with average cost results that are all within 15% of Tabu Search, also when looking at the difference from the best found solution.

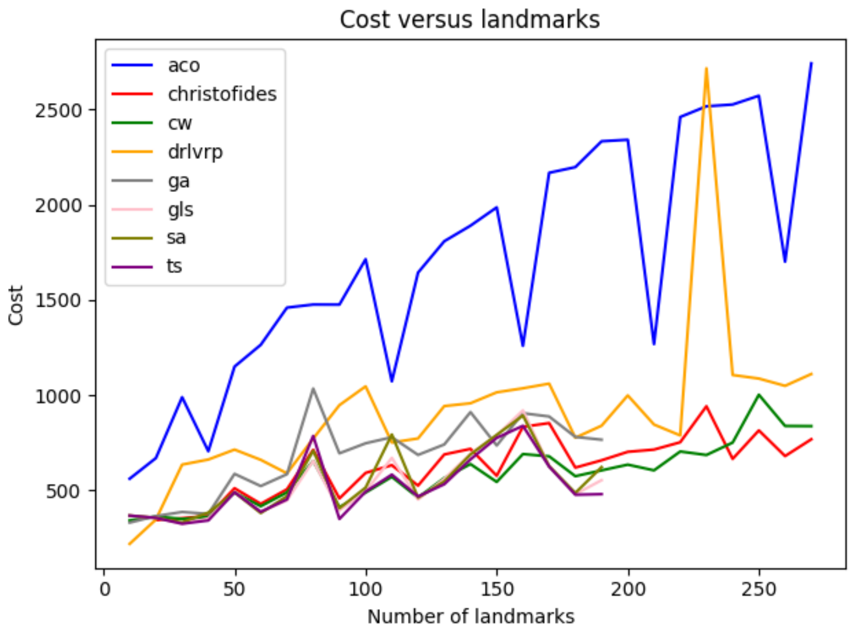


Figure 7.5: Cost measured against an increasing number of landmarks, 10 agents.

7.3 Real-world application

The current system has several elements which allow it to be used as a real-world application. Any solution can be converted back into GPS coordinates, an example of which can be found in Figure 7.8. Moreover, a GUI has been developed for accessible use of the system. An image of the interface can be found in Figure 7.9. As can be seen in this screenshot, the interface allows the user to specify a variety of input settings and load custom files, after which any of the tested algorithms in this research can be used to find a solution and output it to a list of target GPS coordinates. A guide on how to use the GUI in more detail can be found in Appendix E.

Environment	SMT*	DRL*	CW*	CHR	GA	ACO	SA*	TS*	GLS*	Average
bgrid20	2	98	8	20	20	16	5	6	5	20
cgrid10	1	4	3	22	20	26	5	5	5	10
bgrid40		20	7	26	18	16	5	7	5	13
dgrid20		20	7	20	18	12	6	7	7	12
z200		21	16	16	18	22	7	8	8	15
e50		6	5	20	20	22	5	5	5	11
hcl200		96	15	10	20		8	8	8	24
acl50		7	6	8	18		5	5	5	8
i200		25	20	26	20	28	6	6	6	17
l50		9	8	26	16	26	3	4	3	12

Table 7.1: Found optimal number of agents per method. Averages are rounded.
* Number of agents determined by method itself

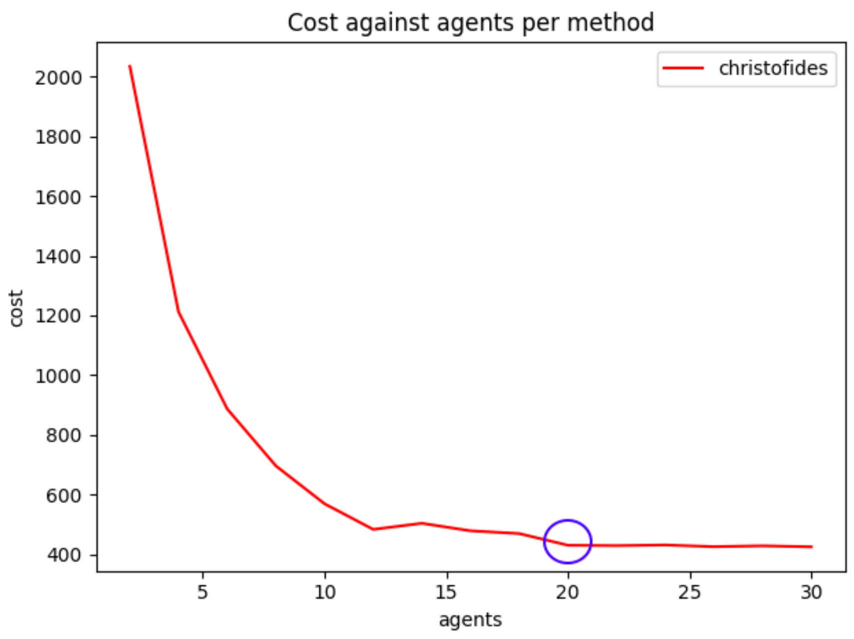


Figure 7.6: Determining the optimal number of agents for Christofides in `bgrid20`

Env.	SMT	DRL	CW	CHR	GA	ACO	SA	TS	GLS
`bgrid20`	253.2	20.3	**4.1**	4.4	1963	95.6	7817	7817	7817
`cgrid10`	65.2	**0.6**	19.8	20.3	1180	135.9	634.8	634.8	634.8
`bgrid40`		**3.1**	5531	5531	21920	6272	75782	75782	75782
`dgrid20`		**1.2**	345.3	345.6	1874	496.2	6442	6442	6442
`z200`		**2.4**	88.5	88.5	7910	622	31338	31338	31338
`e50`		0.6	**0.3**	0.6	1098	51.7	488.6	488.3	488.3
`hcl200`		**3.1**	5.3	5.2	7827		31255	31255	31255
`acl50`		0.6	**0.2**	0.4	980.0		488.2	488.2	488.2
`i200`		**2.4**	7017	7017	14839	8215	38266	38267	38266
`l50`		**0.7**	205.1	205.6	1273	660.3	693.1	693.1	693.1
Avg.	***159.2***	***3.5***	***1322***	***1322***	***6087***	***2069***	***19321***	***19321***	***19321***

Table 7.2: Computation times (s) for each method. Number of agents per method can be found in Table 7.1. The fastest algorithm is shown in **bold**.

Env.	SMT	DRL	CW	CHR	GA	ACO	SA	TS	GLS
`bgrid20`	2508	1931	502.5	**431.1**	597.3	1982	631.5	530.9	587.4
`cgrid10`	824.5	688.0	386.9	**251.3**	259.0	600.4	271.1	265.8	273.0
`bgrid40`		3567	1020	1044	1451	2170	939.0	825.2	**817.1**
`dgrid20`		777.4	376.9	374.7	504.4	919.8	390.0	363.9	**363.7**
`z200`		1120	828.3	816.6	975.2	2747	797.6	**792.0**	800.7
`e50`		468.5	415.6	**329.0**	346.6	601.2	362.2	358.8	363.3
`hcl200`		473.4	296.3	325.7	425.6		295.0	295.3	**294.2**
`acl50`		505.9	284.6	268.9	**258.0**		276.9	275.9	279.2
`i200`		1721	**829.1**	989.8	1282	2732	1013	981.1	925.3
`l50`		1982	1167	**1105**	1166	1305	1128	1114	1168
Avg.	***1666***	***1323***	***610.7***	***593.6***	***724.8***	***1638***	***610.4***	***580.3***	***586.8***

Table 7.3: Cost function results for each method. Number of agents per method can be found in Table 7.1. Best performing algorithm is shown in **bold**.

Env.	SMT	DRL	CW	CHR	GA	ACO	SA	TS	GLS
bgrid20	464%	406%	16%	0%	36%	479%	44%	21%	31%
cgrid10	217%	165%	52%	1%	3%	211%	2%	0%	1%
bgrid40		375%	34%	39%	87%	256%	17%	0%	2%
dgrid20		116%	6%	9%	40%	259%	9%	0%	0%
z200		54%	14%	10%	30%	376%	4%	0%	0%
e50		35%	25%	0%	4%	126%	1%	0%	0%
hcl200		76%	6%	15%	51%		0%	1%	0%
acl50		95%	10%	4%	0%		0%	0%	0%
i200		120%	0%	19%	43%	313%	14%	10%	2%
l50		92%	14%	11%	11%	64%	2%	0%	3%
Avg.	***341%***	***153%***	***18%***	***11%***	***30%***	***261%***	***9%***	***3%***	***4%***

Table 7.4: Percentage difference from the best agent operation time per scenario, where 0% represents the best found total agent operation time.

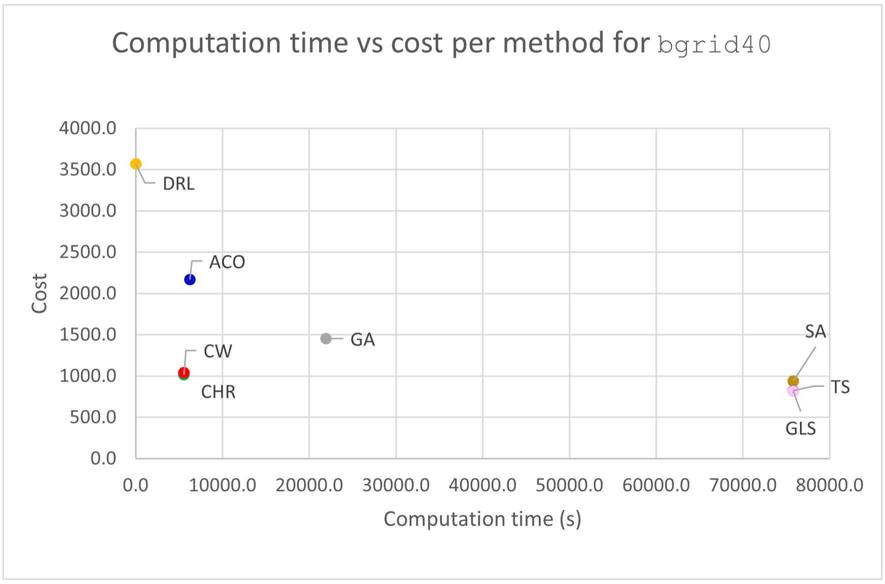
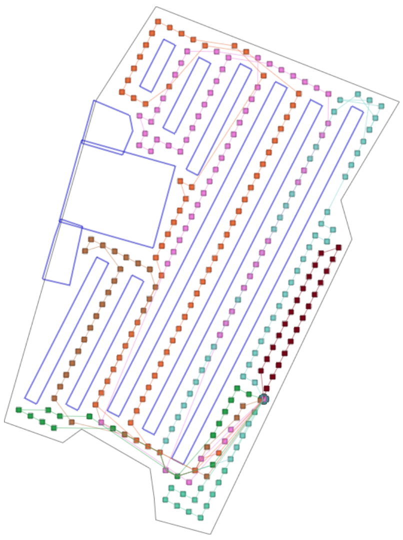
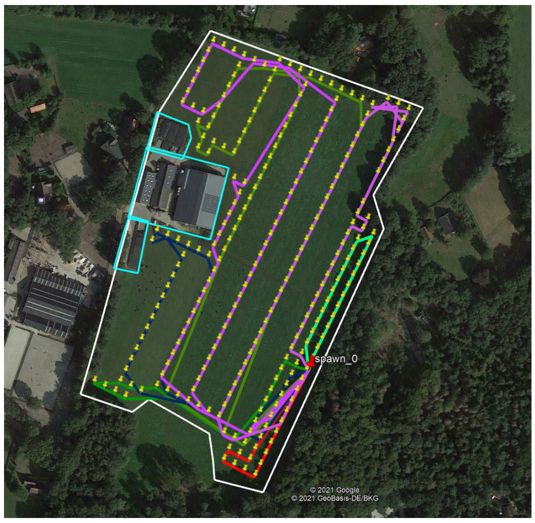


Figure 7.7: Scatter plot of the computation time - cost tradeoff for task bgrid40.

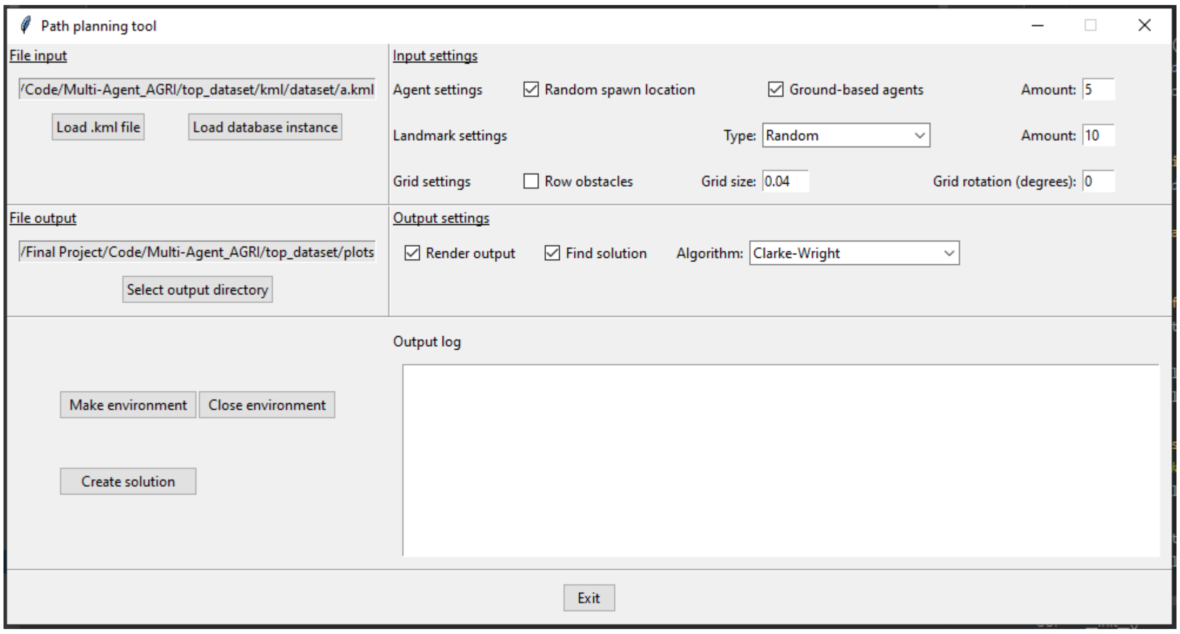


(a) CW solution



(b) CW solution transformed to GPS

Figure 7.8: Solution (CW) to `bgrid40`, both in the simulation environment and in Google Earth after outputting the paths to GPS coordinates.



File input

/Code/Multi-Agent_AGRI/top_dataset/kml/dataset/a.kml

Input settings

Agent settings ☑ Random spawn location ☑ Ground-based agents Amount: 5

Landmark settings Type: Random Amount: 10

Grid settings ☐ Row obstacles Grid size: 0.04 Grid rotation (degrees): 0

File output

/Final Project/Code/Multi-Agent_AGRI/top_dataset/plots

Output settings

☑ Render output ☑ Find solution Algorithm: Clarke-Wright

Output log

Figure 7.9: Screenshot of the Graphical User Interface (GUI)

Chapter 8

Discussion

In this chapter, some points for discussion are given. This includes some discussion on the experiment design, the method and also some discussion about the results from Chapter 7.

8.1 Experiment design

This section raises some discussion about the design of the experiment.

In the current system, a real-world topology consisting of geographic coordinates is transformed to a two-dimensional local coordinate system. However, geographic coordinates cannot be transformed into two dimensions linearly; they take into account the curvature of the earth, making a line between two points an arc rather than a straight line. This is why the haversine formula is used to determine a scaling factor to meters, instead of a standard distance formula such as the Euclidean distance. All this results in the fact that transforming back to geographic coordinates will have a slight error margin, and that the calculated distances also have a slight error margin. For a simple single geographical field this error is negligible (the surface area is almost flat), however for larger topologies this might pose a problem. A way to circumvent this for larger problems would be to calculate all distances using the haversine formula instead of the Euclidean distance, keeping the original latitude-longitude coordinates. In this way all of the discussed approaches should still work and provide a solution that matches the real world more accurately.

8.2 Algorithms

This section includes a few discussion points for the used methods, listed under their respective approach.

Christofides

Currently, k-means clustering is used to transform the VRP into k TSPs, where k is the number of available agents. k-means is one of the clustering methods available which is least susceptible to noise and outliers, and thus produces overall good results for many problems. However, for clustering problems which do not have a circular shape k-means performs worse. In this project, row-like tasks were evaluated, where the best clustering approach would be to group points based on the rows instead of using k-means. Apart from this, investigating other clustering methods for each type of scenarios could result in a more robust Christofides-based solution to the VRP of this project. Moreover, if clustering methods are found to produce robust results, they could be used to transform other promising TSP methods into VRP solutions, such as the deep-learning based TSP solvers discussed in Section 3.3.3.

Clarke-Wright

The sequential approach for Clarke-Wright is not used, only the parallel approach. In some cases, the sequential approach might yield better results at the cost of longer computation times.

OR-Tools metaheuristic approaches

To improve the results for the OR-Tools approaches, there are a few options that can be explored. Firstly, the initial solution is generated each time using the `PATH_CHEAPEST_ARC` setting, while the other settings remain untested. Moreover, the current time limit is based on some initial observations. More extensive research into the optimal time limit for different problem sizes could improve result quality even further. Contrarily, a solution limit can be set to limit the number of explored solutions.

Genetic Algorithm

As with the OR-Tools, the solution limit and time limit for the GA could be improved by performing some more experiments, as the current limits are merely based on some initial observations.

Ant Colony Optimization

As explained by Gaertner and Clark [63], any TSP-related problem needs tailored parameters when applying ACO. The best approach would be to define some general heuristics for each type and size of problem, such that the selected parameters match the problem to some degree. This is not the case in the current implementation, but could be an improvement when using ACO for this type of problem in the future. However, since the results for ACO are significantly worse than the other metaheuristic approaches, it is deemed unlikely that, even with fine-tuned parameters, the high accuracy of these other approaches can be matched.

As explained previously, it is not possible with the current system to produce results for ACO for scenarios in which the points are clustered. This problem can likely be avoided by adjusting some of the parameters or the construction of the visibility matrix, which has not been done in this research.

POPCORN

The current implementation of POPCORN does not use the pre-selected landmarks, but instead creates its own grid for CPP. This results in the fact that POPCORN solutions are not one-on-one comparable to the actual aerial grid-based CPP tasks. However, the amount of landmarks generated by POPCORN is similar to the actual value (167 instead of 126 for `bgrid20` and 60 instead of 54 `cgrid10`). Moreover, the grid size matches exactly. Nonetheless, for the best one-on-one comparison with other algorithms the grid used for POPCORN should be constructed such that it matches the predefined landmarks exactly.

Mean Field RL

Unfortunately, the current approach for using mean field MARL did not result in a converged cost result within the project's time limit. However, this approach, as well as the other discussed MARL approaches are still seen as promising methods that could provide a new type of solution to vehicle routing problems.

There are various reasons that could cause the current approach to not converge. The DQN structure could not be deep enough for the problem. Moreover, the observation and rewards of the agents might not provide sufficient information for the agents to learn from. It could also be that simply more training is necessary. Lastly, something to consider is the decay function of ϵ, which could allow for better initial exploration during training.

RL-VRP

The models for RL-VRP cannot be assumed to produce on average the most optimal solution for their respective problem sizes. The scores on the test set (see Figure 6.3) are worse than the scores reported by the method's creators [50], although for the smaller problem sizes the scores are close. As can be seen in the graph, the test score does not increase linearly. Longer training, perhaps with some random resets could solve this problem. Also, the demand constraints used during training could be chosen such that they match the problem demand more closely. For an explanation of how the demand was included as a part of the problem, see 6.3.

As another discussion point for the RL-VRP approach, its solutions are presented also for the problems with obstacles, but the method does not take those into account. Thus, all paths between landmarks are considered straight edges, which makes the solutions for those problems far from optimal. The boundary of the environment is also not taken into account. Added some way of feeding this data to the network would improve the algorithm's performance on concave topologies and topologies with obstacles, for example adding some convolutional layers to the network and passing an image of the environment with obstacles as input.

Lastly, better results could be obtained with RL-VRP when using the beam search approach instead of the greedy approach. This increases training times and complexity, but could improve overall model performance.

8.3 Results

This section contains some discussion and takeaways about the results.

Number of available agents

Currently, used the cost function includes only two main factors: fairness between the agents and the total operation time. However, as can be seen with for example Christofides and RL-VRP, sometimes the best solution (cost-wise) includes a large number of agents, which often might not be available for the task. It is assumed that medium-scale farmers will often not have more than 5 to 10 agents available. Of course, there is the option of combining agent routes until the number of routes matches the number of available agents, as described in Section 5.1.3, or of selecting a method which uses less agents at a still acceptable cost.

To better reflect on the best methods found and feasibility of the number of agents, the overview in Table 8.1 is created. In the left side of this table, the best algorithm is reported for the case that only 10 agents are available. On the right, the best algorithm is shown for the case that any number of agents can be used.

$m \leq 10$	Accurate	m	Fastest	m	$m \leq \infty$	Accurate	m	Fastest	m
`bgrid20`	CW	8	CW	8		CHR	20	CHR	20
`cgrid10`	TS	5	CHR	10		CHR	22	CHR	22
`bgrid40`	GLS	5	CW	7					
`dgrid20`	GLS	7	CW	7					
`z200`	TS	8	TS	8				CW	16
`e50`	TS	5	CHR	10		CHR	20	CHR	20
`hcl200`	GLS	8	CHR	10					
`acl50`	CHR	8	CW	6		GA	18		
`i200`	CW	10*	CW	10*		CW	20	CW	20
`l50`	TS	4	CW	8		CHR	26		

Table 8.1: Best scoring algorithms per scenario. The left table is limited to 10 agents, and on the right are possible improvements with larger m (number of agents). The cost of the fastest algorithm should be within 5% of the most accurate result.
*Clarke-Wright with shortest routes combined until 10 routes remain.

Cost function

As explained in Section 6.1, the balancing factors q and z are set to 0.5 for all experiments. They could be changed to better fit the scenario during the experiments. For example, for aerial robots the total flight time is more important than the distance division between agents, which could be reflected by making z larger than 0.5. This is something to take into account in future experiments.

Demand constraint

Some of the algorithms tested solve the capacitated vehicle routing problem (CVRP), which means that they also solve for a demand constraint; each landmark is a customer and a vehicle can supply a limited number of these customers before returning to the depot. In most cases, the demand of each landmark was set to 0 such that each vehicle can supply an infinite number of customers. The demand could be used to get a more equal distribution between agents. The idea is that each customer has a demand such that a vehicle can visit $1/\phi$ times the *expected* number of customers on its route, with ϕ a constant smaller than or equal to one, n the number of customers and m the number of vehicles. The following formula is used to determine the demand of each node, where each vehicle has a capacity of 1:

$$demand_n = \phi\frac{m}{n}$$

To recreate the situation used in the results in Chapter 7, ϕ is simply equal to 0, except for GA and RL-VRP, which use a ϕ of 1 as explained in Section 6.3. The result for different values of ϕ (phi) for the Clarke-Wright algorithm is shown in Figure 8.1. As can be seen, there is a decrease in cost for larger values of ϕ.

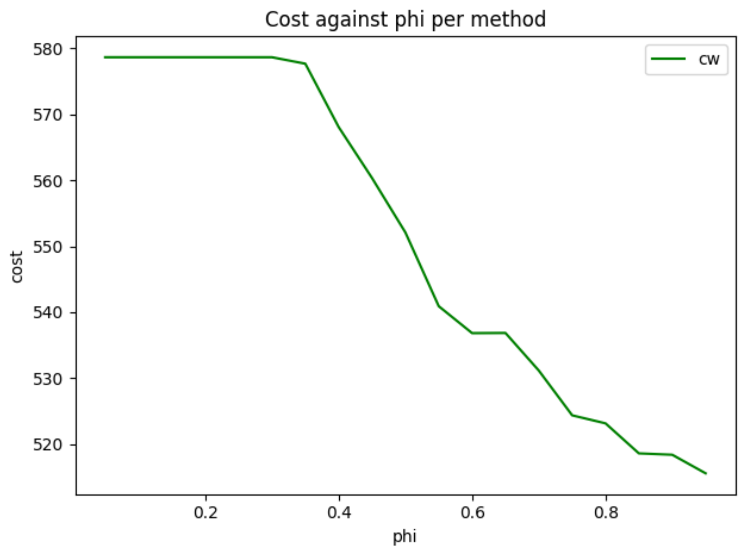


Figure 8.1: Resulting cost for Clarke-Wright on the `bgrid40` scenario, for different values of ϕ. 20 agents are available for each experiment.

Chapter 9

Conclusion and recommendations

In this chapter, the research questions will be answered in the form of a conclusion. Moreover, some recommendations for future work are provided.

9.1 Conclusion

To answer the main research question of this research, first the subquestions will be answered.The questions are discussed in the order they are presented in Chapter 1.

- *How can be ensured that the system scales well with regard to the number of agents in the swarm?*

Ensuring proper scaling can be done through selecting the right method for the problem size. Thus far, the only method which scales badly with regards to the number of agents is ACO (see Figure 7.1), which makes it unsuitable for situations with larger numbers of agents. For other methods, scaling problems do not occur with larger numbers of agents, but with larger numbers of landmarks (see Figure 7.2). Time-limited iterative approaches such as the Genetic Algorithm and OR-Tools approaches are unable to converge to a good solution within a small time limit for problems with larger number of landmarks. Christofides scales well in both regards, and Clarke-Wright does not show any scaling issues either.

- *Which approach is best suited for which individual topologies and/or combinations of topologies?*

Looking at the results discussed in Section 7.2, the method that performs the best on average is Christofides. However, to obtain these results, a large number of agents had to be deployed, as discussed before. Looking at the agent operation time savings in Table 7.4, Clarke-Wright is a close second in terms of cost. Both of these two methods are not as fast in terms of computation time as RL-VRP, however the scores for RL-VRP are too bad to compete with CW and Christofides. POPCORN is slow and results in high-cost solutions, and the same holds for ACO. The three OR-Tools approaches (SA, TS, and GLS) result in the best achieved scores for some of the topologies, and are expected to score even better given more computation time budget.

There is always a tradeoff between computation time and result quality when comparing these types of algorithms. If computation time is the bottleneck, and fast solutions are necessary (for example for an environment which changes during operation, requiring recalculations), Christofides appears to be the best method if a large number of agents is available. If this is not the case, Clarke-Wright might be a better option. However, if calculations are done beforehand, and more computation time budget is available, one of the OR-Tools metaheuristics is the better option, with Tabu Search showing the best cost results. Based on how the methods scale, the metaheuristic approaches are better suited for smaller problems, since they are likely to produce better results within a reasonable amount of time. For larger problems, the amount of time needed to find a better solution than CW or Christofides might become infeasible.

Looking at which specific topology requires which method, it is hard to make a clear distinction between the scenarios. The problem size seems to impact the performance of the methods much more than the type of task at hand. This means that most of the methods presented are flexible enough to cover a wide range of problems. For a quick overview of the best method per topologies for different numbers of available agents, Table 8.1 can be consulted.

- *How to simulate a realistic scenario of a robotic swarm in an agricultural field, taking into account the robots' constraints and limitations?*

A realistic simulation scenario is constructed for this research, using geographical coordinates of actual agricultural topologies. Row-like crops are simulated by adding rectangular obstacles. The robot's constraints are modeled using their speed to find the maximum distance they can travel before their battery is depleted. However, there are still quite some limitations to the simulations which should be removed if the simulations are to match the real world more closely. This can be done by relaxing some of the constraints, such as taking into account the time certain tasks take or that the agent is not able to travel at full speed at all times. For more ideas on which assumptions to relax, see 9.2.

Main conclusion

With these answered subquestions, now the main research question can be addressed:

> ***How to design a system for efficient path planning and target assignment for robotic swarms in agricultural applications?***

In this research, a system is designed. Currently, it is mainly used to evaluate the performance of different path planning methods to find the most suitable method for different jobs. A more finalized version of this system contains a number of path planning algorithms and selects the most suitable approach based on the topology size, the number of available agents and the computation time available. To keep computation times reasonable, any solution produced by the system will often not be the optimal solution, however it will be efficient enough for the specified job.

The system is transformed into an accessible real-world application through the creation of a Graphical User Interface (GUI). The system contains many useful features, such as for example obstacle avoidance and generation of row-like structures. User input from Google Earth is used, and the system output can be plotted into Google Earth as well. The output solution can be transformed into GPS coordinates which can be used by the robots directly.

The system meets all specified *must have* and *should have* requirements from Section 4.2, and it also meets one of the *could have* requirements: the GUI.

9.2 Recommendations

There are various recommendations for continuing this research. Some concern improving the solution quality for a better system output, others discuss extending the tested methods with untested promising approaches. Moreover, some recommendations are given for extending the system functionalities, for example by relaxing some of the assumptions. Apart from these recommendations, it would be good to keep an eye on the robotics business sector, since deploying robot swarms in agriculture is in development at several companies, for a wide range of applications. For example, it was announced recently that drones are being developed for fruit picking jobs in orchards [64]. Projects like these can provide useful insights into the relevance of this type of research and possible applications and solutions.

Improving solution quality

The solution quality of the system could be improved in various ways. If there is enough computation time available, the individual agent routes could be improved (or verified) using an exact TSP solver such as Concorde [17]. Moreover, adding a demand constraint could be used to increase fairness between agents, as discussed in Section 8.3.

To verify the output solution quality, the methods included in the final system could be used to produce results to know VRP datasets, for example the VRP-REP database instances[1], or the instances tested on in the surveyed literature, for example the problems evaluated by Osman [29] in his comparison.

Other promising algorithms

In this research, the use of multi-agent reinforcement learning (MARL) approaches is investigated. However, applying them in practice has thus far been unsuccessful. It is recommended to further explore this group of algorithms for the topic of path planning, and the literature supplied in this thesis might be a good starting point for that.

Another largely unexplored method in this thesis is the DeepWay coverage path planning approach [52], which is a promising method to look into for the case of row-aligned crop fields.

Using deep learning and reinforcement learning for combinatorial optimization problems such as the VRP is an active field of research at the moment. It is likely that, in the near future, a method will become available which fits this problem globally and is able to produce solid solutions at very low computation (inference) times.

Extending system functionality

To add more functionalities to the system, the type of VRP used can be adjusted. For example, a VRP with time windows can be used to represent specific times at which crops have to be inspected. Apart from the demand constraint discussed earlier, making the VRP a CVRP could account for an actual agent capacity and delivery load, for example to represent the amount of pesticide which a drone can carry.

Another way of extending functionality would be relaxing some of the assumptions made in Section 4.3. This would result in a system taking into account more real-world factors. For example, if the VRP would be asymmetric, wind direction or slopes could be taking into account when calculating the distance tables. Another example would be to add start-up or takeoff times, and service times at at each point of interest, such that the agent is no longer traveling at full speed the whole time. Also, the distance from the controller could be another constraint to add to the case of aerial agents.

[1]Available from `http://www.vrp-rep.org/`

Bibliography

[1] “United Nations, Department of Economic and Social Affairs: How certain are the United Nations global population projections?” accessed: 2020-11-19. [Online]. Available: https://www.un.org/en/development/desa/population/publications/pdf/popfacts/PopFacts_2019-6.pdf

[2] Y. Edan, S. Han, and N. Kondo, “Automation in agriculture,” in *Springer handbook of automation*. Springer, 2009, pp. 1095–1128.

[3] L. A. Garibaldi, B. Gemmill-Herren, R. D’Annolfo, B. E. Graeub, S. A. Cunningham, and T. D. Breeze, “Farming approaches for greater biodiversity, livelihoods, and food security,” *Trends in ecology & evolution*, vol. 32, no. 1, pp. 68–80, 2017.

[4] T. Batey, “Soil compaction and soil management–a review,” *Soil use and management*, vol. 25, no. 4, pp. 335–345, 2009.

[5] “Hortibiz Daily: Are light autonomous robots the answer to soil compaction?” accessed: 2021-03-08. [Online]. Available: https://www.hortibiz.com/news/?tx_news_pi1%5Bnews%5D=37895&cHash=89db2e88c1be0c8d810383c09c2c8553

[6] J. V. Stafford, “Implementing precision agriculture in the 21st century,” *Journal of Agricultural Engineering Research*, vol. 76, no. 3, pp. 267–275, 2000.

[7] D. Albani, J. IJsselmuiden, R. Haken, and V. Trianni, “Monitoring and mapping with robot swarms for agricultural applications,” in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2017, pp. 1–6.

[8] “Roboticsbiz: Robotics in agriculture: advantages and disadvantages,” accessed: 2021-03-08. [Online]. Available: https://roboticsbiz.com/robotics-in-agriculture-advantages-and-disadvantages/

[9] N. Botteghi, A. Kamilaris, L. Sinai, and B. Sirmacek, “Multi-agent path planning of robotic swarms in agricultural fields,” in *2020 24th ISPRS Congress on Technical Commission I*, vol. 5, no. 1. Copernicus GmbH, 2020, pp. 361–368.

[10] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, “A compilation of UAV applications for precision agriculture,” *Computer Networks*, vol. 172, p. 107148, 2020.

[11] “Croptracker blog: Drone Technology in Agriculture,” accessed: 2020-11-12. [Online]. Available: https://www.croptracker.com/blog/drone-technology-in-agriculture.html

[12] T. Oksanen and A. Visala, “Coverage path planning algorithms for agricultural field machines,” *Journal of field robotics*, vol. 26, no. 8, pp. 651–668, 2009.

[13] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” *Robotics and Autonomous systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[14] K. Shah, G. Ballard, A. Schmidt, and M. Schwager, “Multidrone aerial surveys of penguin colonies in antarctica,” *Science Robotics*, vol. 5, no. 47, 2020.

[15] H. Seyyedhasani and J. S. Dvorak, “Using the vehicle routing problem to reduce field completion times with multiple machines,” *Computers and Electronics in Agriculture*, vol. 134, pp. 142–150, 2017.

[16] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” *Management science*, vol. 6, no. 1, pp. 80–91, 1959.

[17] “Concorde TSP solver homepage,” accessed: 2021-03-08. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/index.html

[18] D. Applegate, W. Cook, S. Dash, and A. Rohe, “Solution of a min-max vehicle routing problem,” *INFORMS Journal on computing*, vol. 14, no. 2, pp. 132–143, 2002.

[19] “Wikipedia: Nearest neighbour algorithm,” accessed: 2021-03-11. [Online]. Available: https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

[20] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II, “An analysis of several heuristics for the traveling salesman problem,” *SIAM journal on computing*, vol. 6, no. 3, pp. 563–581, 1977.

[21] N. Christofides, “Worst-case analysis of a new heuristic for the travelling salesman problem,” Carnegie-Mellon University, Pittsburgh, PA Management Sciences Research Group, Tech. Rep., 1976.

[22] “Wikipedia: 2-opt,” accessed: 2021-03-11. [Online]. Available: https://en.wikipedia.org/wiki/2-opt

[23] “Wikipedia: 3-opt,” accessed: 2021-03-11. [Online]. Available: https://en.wikipedia.org/wiki/3-opt

[24] S. Lin and B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[25] K. Helsgaun, “An effective implementation of the lin-kernighan traveling salesman heuristic,” *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[26] G. Clarke and J. W. Wright, “Scheduling of vehicles from a central depot to a number of delivery points,” *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[27] J. Lysgaard, “Clarke & wright’s savings algorithm,” *Department of Management Science and Logistics, The Aarhus School of Business*, vol. 44, 1997.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[29] I. H. Osman, “Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem,” *Annals of operations research*, vol. 41, no. 4, pp. 421–451, 1993.

[30] M. Gendreau, A. Hertz, and G. Laporte, “A tabu search heuristic for the vehicle routing problem,” *Management science*, vol. 40, no. 10, pp. 1276–1290, 1994.

[31] C. Voudouris and E. P. Tsang, “Guided local search,” in *Handbook of metaheuristics*. Springer, 2003, pp. 185–218.

[32] “Google OR-Tools,” accessed: 2021-03-10. [Online]. Available: https://developers.google.com/optimization

[33] B. M. Baker and M. Ayechew, “A genetic algorithm for the vehicle routing problem,” *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, 2003.

[34] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, “A hybrid genetic algorithm for multidepot and periodic vehicle routing problems,” *Operations Research*, vol. 60, no. 3, pp. 611–624, 2012.

[35] J. E. Bell and P. R. McMullen, “Ant colony optimization techniques for the vehicle routing problem,” *Advanced engineering informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[36] B. Yu, Z.-Z. Yang, and B. Yao, “An improved ant colony optimization for vehicle routing problem,” *European journal of operational research*, vol. 196, no. 1, pp. 171–176, 2009.

[37] Y. Marinakis and M. Marinaki, “A hybrid genetic–particle swarm optimization algorithm for the vehicle routing problem,” *Expert Systems with Applications*, vol. 37, no. 2, pp. 1446–1455, 2010.

[38] T. J. Ai and V. Kachitvichyanukul, “A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery,” *Computers & Operations Research*, vol. 36, no. 5, pp. 1693–1702, 2009.

[39] S. MirHassani and N. Abolghasemi, “A particle swarm optimization algorithm for open vehicle routing problem,” *Expert Systems with Applications*, vol. 38, no. 9, pp. 11 547–11 551, 2011.

[40] H. R. Beom and H. S. Cho, “A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning,” *IEEE transactions on Systems, Man, and Cybernetics*, vol. 25, no. 3, pp. 464–477, 1995.

[41] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

[42] H. Qie, D. Shi, T. Shen, X. Xu, Y. Li, and L. Wang, “Joint optimization of multi-uav target assignment and path planning based on multi-agent reinforcement learning,” *IEEE Access*, vol. 7, pp. 146 264–146 272, 2019.

[43] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” *arXiv preprint arXiv:1705.08926*, 2017.

[44] S. Iqbal and F. Sha, “Actor-attention-critic for multi-agent reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 2961–2970.

[45] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, “Mean field multi-agent reinforcement learning,” *arXiv preprint arXiv:1802.05438*, 2018.

[46] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, “Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning,” in *The World Wide Web Conference*, 2019, pp. 983–994.

[47] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson, “Learning to communicate with deep multi-agent reinforcement learning,” in *Advances in neural information processing systems*, 2016, pp. 2137–2145.

[48] W. Kim, M. Cho, and Y. Sung, “Message-dropout: An efficient training method for multi-agent deep reinforcement learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6079–6086.

[49] B. Zhang, Z. Mao, W. Liu, and J. Liu, “Geometric reinforcement learning for path planning of uavs,” *Journal of Intelligent & Robotic Systems*, vol. 77, no. 2, pp. 391–409, 2015.

[50] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác, “Reinforcement learning for solving the vehicle routing problem,” in *Advances in Neural Information Processing Systems*, 2018, pp. 9839–9849.

[51] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” *arXiv preprint arXiv:1611.09940*, 2016.

[52] V. Mazzia, F. Salvetti, D. Aghi, and M. Chiaberge, “Deepway: a deep learning estimator for unmanned ground vehicle global path planning,” *arXiv preprint arXiv:2010.16322*, 2020.

[53] X. Bresson and T. Laurent, “The transformer network for the traveling salesman problem,” *arXiv preprint arXiv:2103.03012*, 2021.

[54] Y. Kaempfer and L. Wolf, “Learning the multiple traveling salesmen problem with permutation invariant pooling networks,” *arXiv preprint arXiv:1803.09621*, 2018.

[55] S. X. Yang and C. Luo, “A neural network approach to complete coverage path planning,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 718–724, 2004.

[56] N. Noguchi and H. Terao, “Path planning of an agricultural mobile robot by neural network and genetic algorithm,” *Computers and electronics in agriculture*, vol. 18, no. 2-3, pp. 187–204, 1997.

[57] “Wikipedia: MoSCoW method,” accessed: 2021-04-26. [Online]. Available: https://en.wikipedia.org/wiki/MoSCoW_method

[58] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[59] “Wikipedia: A* Search,” accessed: 2021-03-11. [Online]. Available: https://en.wikipedia.org/wiki/A*_search_algorithm

[60] "Wikipedia: k-means clustering," accessed: 2021-03-11. [Online]. Available: https://en.wikipedia.org/wiki/K-means_clustering

[61] W. Liu, S. Li, F. Zhao, and A. Zheng, "An ant colony optimization algorithm for the multiple traveling salesmen problem," in *2009 4th IEEE conference on industrial electronics and applications*. IEEE, 2009, pp. 1533–1537.

[62] M. Dorigo and T. Stützle, "Ant colony optimization: overview and recent advances," *Handbook of metaheuristics*, pp. 311–351, 2019.

[63] D. Gaertner and K. L. Clark, "On optimal parameters for ant colony optimization algorithms." in *IC-AI*. Citeseer, 2005, pp. 83–89.

[64] "Fresh Plaza: Kubota invests in Tevel, flying autonomous fruit-picking robots," accessed: 2021-03-24. [Online]. Available: https://www.freshplaza.com/article/9288636/kubota-invests-in-tevel-flying-autonomous-fruit-picking-robots/

Appendix A

Python packages and code repositories

The packages and repositories for each method and the base environment can be found in Table A.1. The other packages used, which can also be found in the `requirements.txt` file of the project, are as follows:

- `gym` (0.10.5)
- `numpy` (1.19.5)
- `pyglet` (1.5.0)
- `scikit-learn` (0.23.2)
- `shapely` (1.7.1)
- `torch` (1.6.0)

Packages used for making plots and outputting the Google Earth visualization:

- `cycler` (0.10.0)
- `matplotlib` (3.3.2)
- `pandas` (1.2.1)
- `simplekml` (1.3.2)

Each of these packages can be installed separately using `pip install (...)`, or all at once using `pip install -r requirements.txt`. Note that `wadl-planner` requires at least Python 3.8. The used `wadl-planner` version is 1.0.1, and the used `ortools` version is 8.2.0.
`tensorflow` 2.3.0 was used to test the Mean Field approach, however it is not used in the rest of the project. When using Anaconda, it can be installed with `conda install tensorflow`.

Environment		**Available**
Multi-Agent Particle Environment		Github (`openai`[a])

Method	**Reference**	**Available**
Christofides	[21]	Github (`Retsediv`[b])
SA, TS, GLS	[32]	`pip install ortools`
POPCORN	[14]	`pip install wadl-planner`
RL-VRP	[50]	Github (`mveres01`[c])
ACO	[9] [61] [62]	From previous project
CW, GA	[27] [33]	Programmed based on references
Mean Field Q-Learning	[45]	Github (`mlii`[d])

Table A.1: Used repositories and packages for the tested algorithms and base environment.

[a]Available from `https://github.com/openai/multiagent-particle-envs`
[b]Available from `https://github.com/Retsediv/ChristofidesAlgorithm`
[c]Available from `https://github.com/mveres01/pytorch-drl4vrp`
[d]Available from `https://github.com/mlii/mfrl`

Appendix B

Generating simulation environments from Google Earth

This is a step-by-step guide of how to create a topology in Google Earth for use with the simulation environment.

B.0.1 Step-by-step process

1. Navigate to the area you're interested in in Google Earth Pro[1] (desktop version, free to download).
2. Create a new folder.
3. Add any number of Polygons and Placemarks on the map, and store them in your folder. An overview of which objects to create can be found below.
4. Make sure your topology has at least one *boundary* polygon and at least one *agent placemark.*
5. Right-click your folder and select Save as...
6. Save the file as a `.kml` file.
7. You can now import the topology as a simulation world by passing the file path to the .kml file as the top argument when creating the `Topology` class for the system.

[1] Available from `https://www.google.com/earth/versions`

B.0.2 Object options

Boundary

Only one instance allowed. Adding a boundary is obligatory.

Create a polygon to mark the simulation environment's boundary. Make sure its name contains ”boundary”. Can be either convex or concave.

Obstacle / Wall

Any number of instances allowed.

Create a polygon to mark an obstacle or wall. Obstacles should be convex polygons. If they are not convex, their convex hull will be imported instead when creating the environment.

Add the word **”hard”** to an obstacle's name to make sure it is not possible for drones to fly over the obstacle. If it is *not* **”hard”**, ground agents will collide with it, but drones will be able to fly over it. Besides this, the obstacle's name does not matter, but it should *not* contain **”boundary”**.

Agent / Agent spawn location

Any number of instances allowed. At least one instance is obligatory for correct functioning of the simulator.

Create a placemark to either mark the starting spot for a single agent, or a group of agents. Its name should contain the word **”agent”**.

Landmark / point of interest

Any number of instances allowed.

Create a placemark to mark a point of interest. The point's name does not matter, although it should not contain the word **”agent”**.

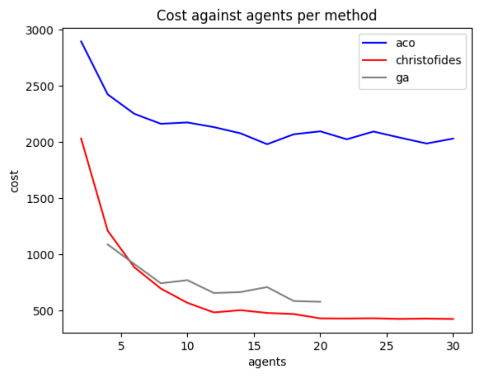
B.0.3 Other tips

If you set a polygon area to *outlined* instead of *filled* or *filled+outlined*, it is much easier to work with.

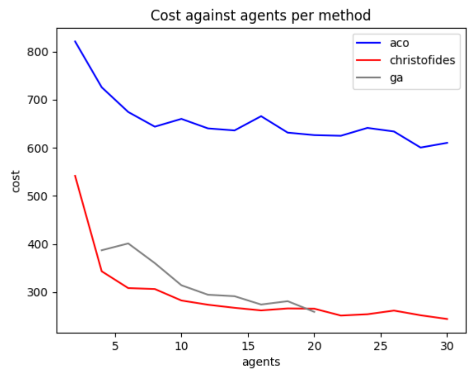
Appendix C

Agent optimization plots

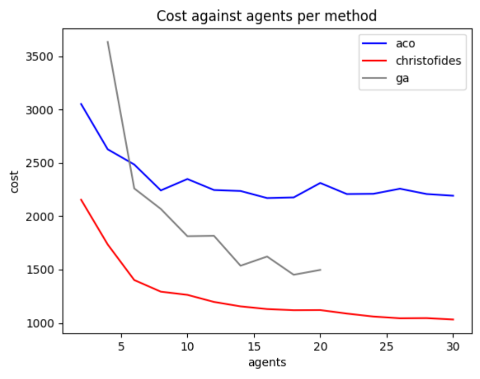
In this appendix, the plots used to find the optimal number of agents per scenario are shown for reference. See Figure C.1 and Figure C.2.



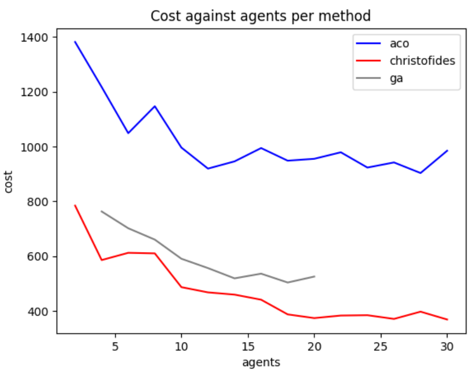
(a) `bgrid20`



(b) `cgrid10`

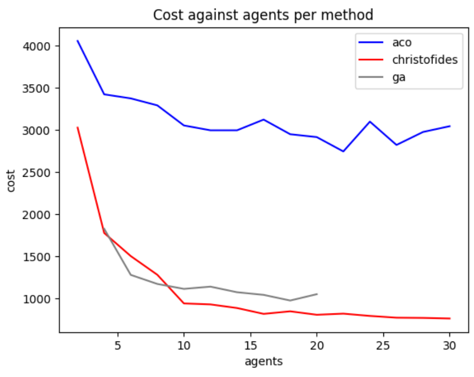


(c) `bgrid40`

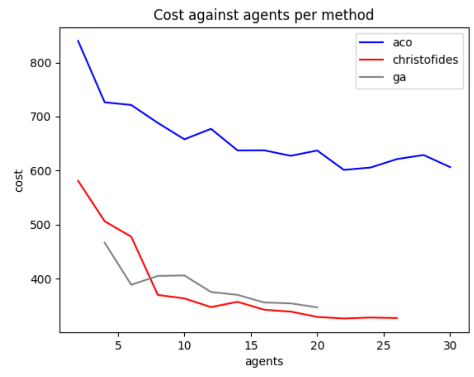


(d) `dgrid20`

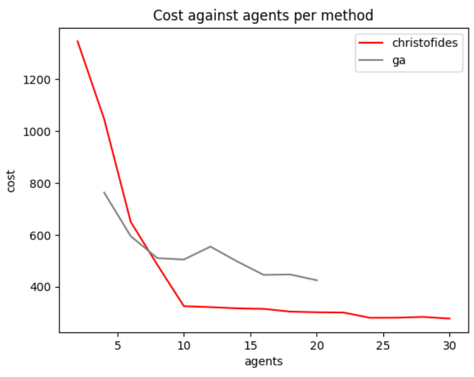
Figure C.1: Optimal agent plots, part 1



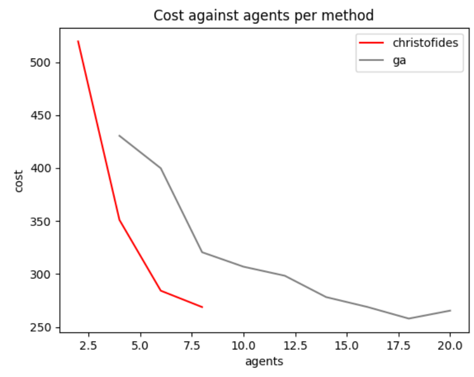
(a) `z200`



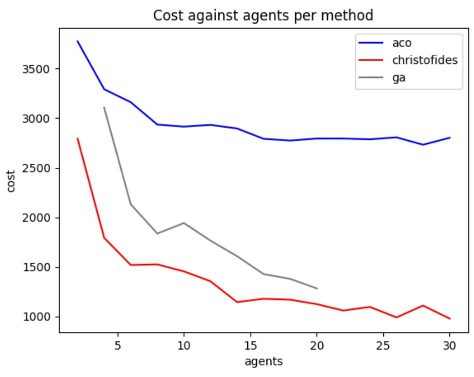
(b) `e50`



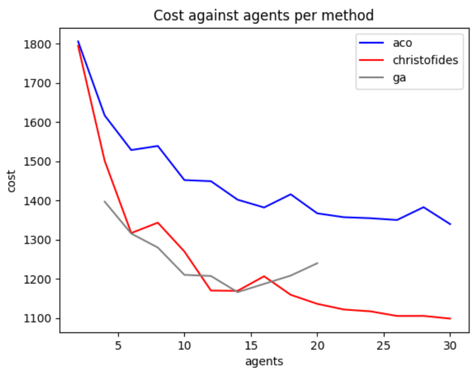
(c) `hcl200`



(d) `acl50`



(e) `i200`



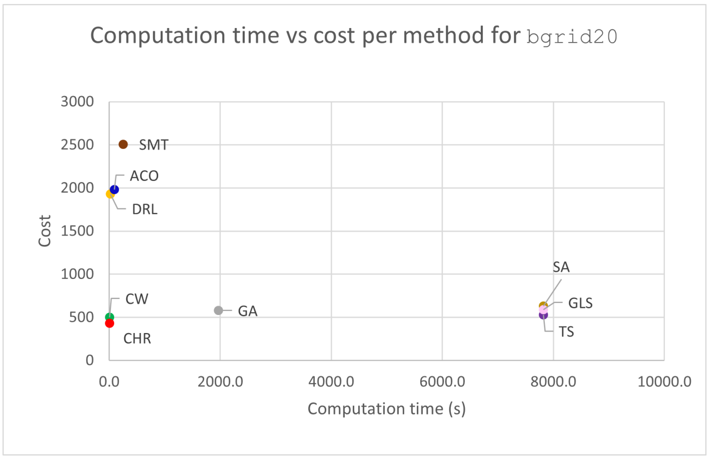
(f) `l50`

Figure C.2: Optimal agent plots, part 2

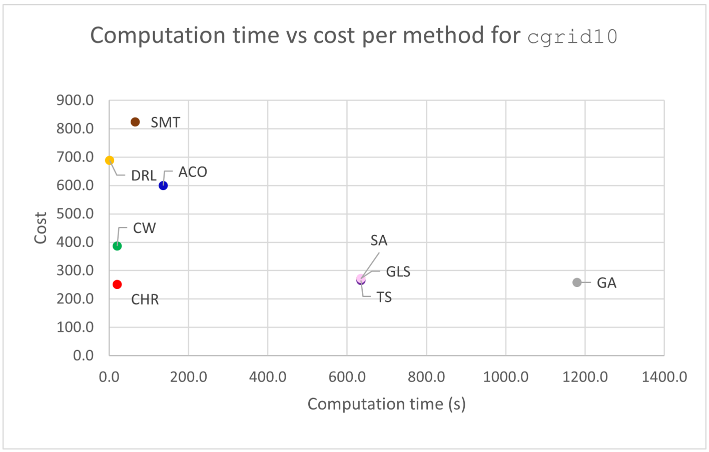
Appendix D

Cost - computation time plots

In this appendix, a scatter plot is presented for each scenario. In each of these plots, the cost is plotted against the computation time for each method that is used to find a solution.

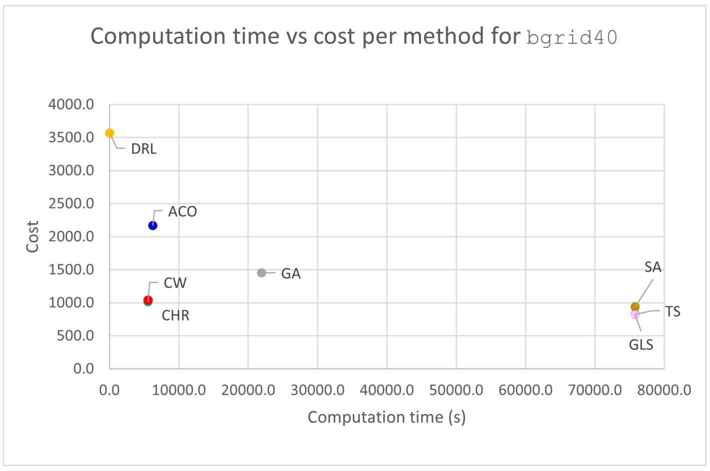


(a) `bgrid20`

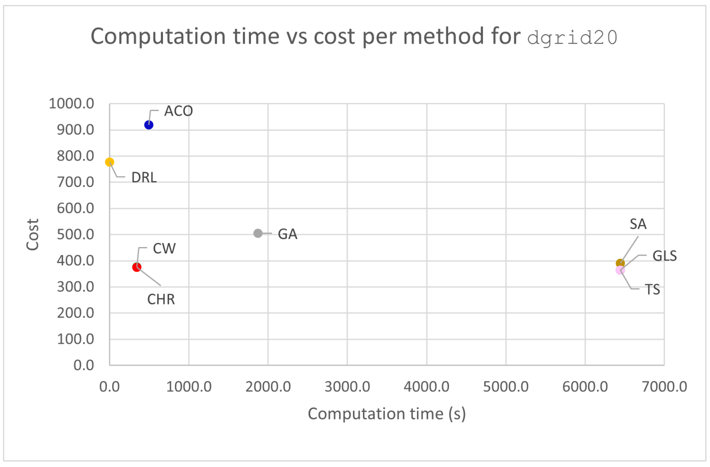


(b) `cgrid10`

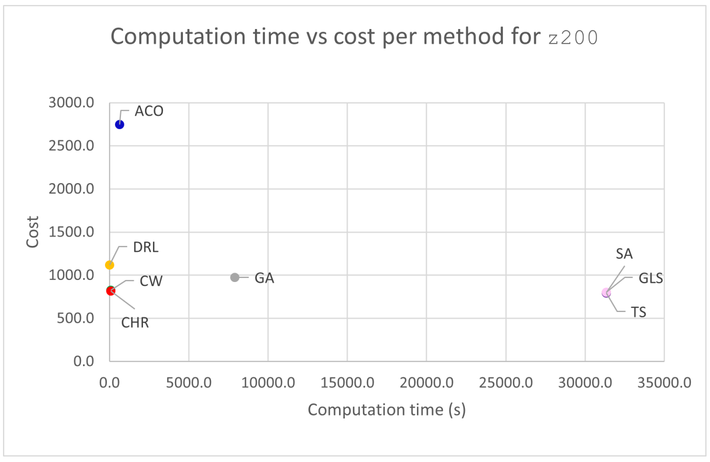
Figure D.1: Cost - computation time tradeoff scatter plots, part 1



(a) `bgrid40`

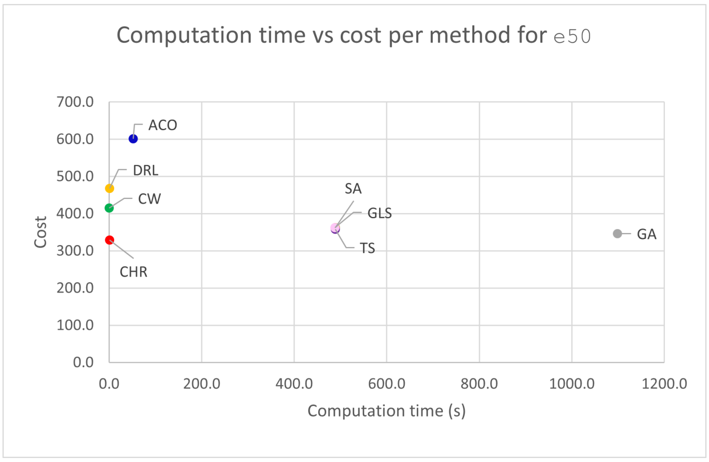


(b) `dgrid20`

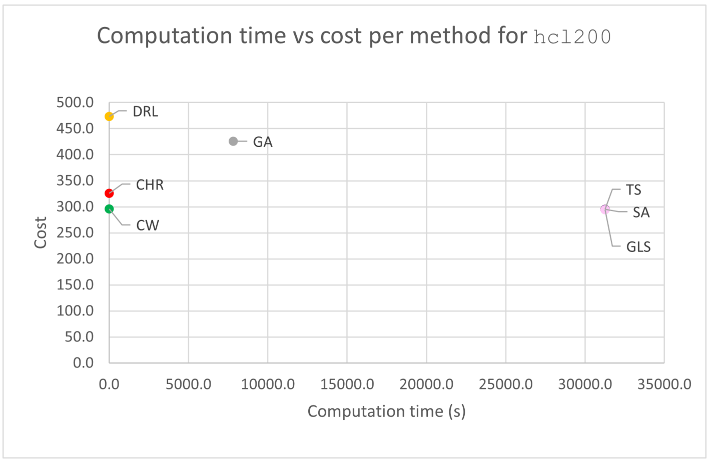


(c) `z200`

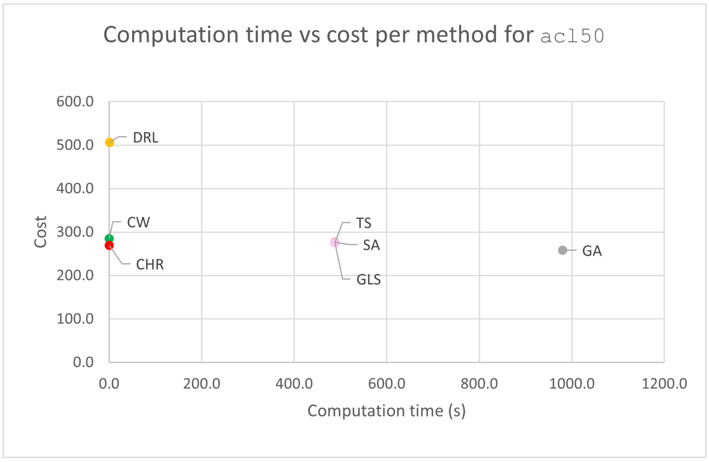
Figure D.2: Cost - computation time tradeoff scatter plots, part 2



(a) e50

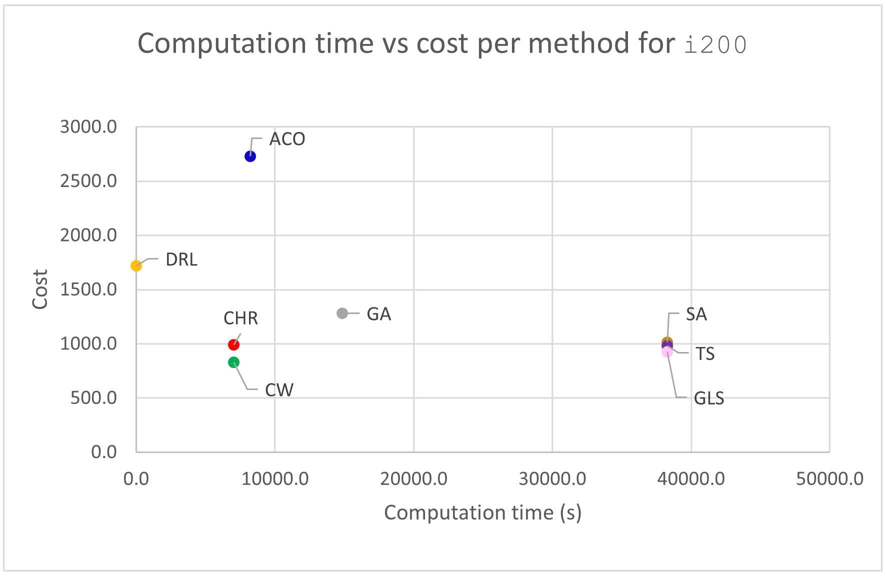


(b) hcl200

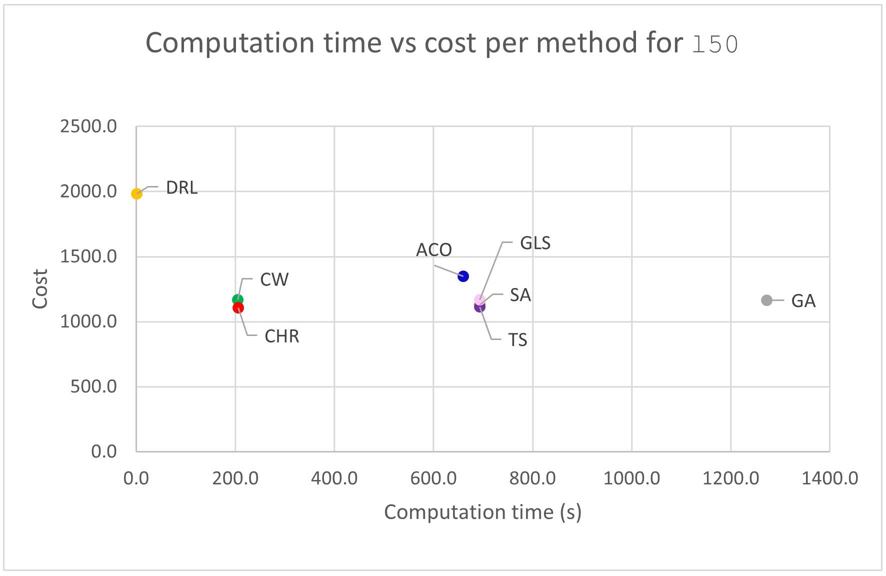


(c) acl50

Figure D.3: Cost - computation time tradeoff scatter plots, part 3



(a) i200



(b) 150

Figure D.4: Cost - computation time tradeoff scatter plots, part 4

Appendix E

Usage guide for the Graphical User Interface

E.0.1 Getting the GUI

The GUI for the path planning tool and installation instructions can can be found at `https://github.com/jmpostmes/Multi-Agent_GPS_Planning_Tool`

E.0.2 Usage guide

Press ‘Exit‘ to exit the tool.

File input

As input, there are two options to choose from: load a Google Earth `kml` file or load a premade scenario from file. For instructions how to create a topology in Google Earth for use with the tool, see Appendix B

To load an earlier stored scenario, select either the `scenario.dictionary`, `topology.dictionary` or `world.dictionary` file for loading. The others will be included automatically.

After choosing a file to load, press the `make environment` button. An environment for planning will now be created with the settings from **Input settings**. If loading a premade scenario, **Input settings** will be updated to match the settings of the loaded scenario.

File output

To select a custom output folder, click `Select output directory` and select the folder where you want the output of the tool to be stored. By default, this is the root folder containing the tool files.

Input settings

There are various input settings that can be set when creating an environment.

`Random spawn location` randomly relocates the agent starting location if set.

`Ground-based agents` sets the agents to aerial agents when unchecked, and ground agents when checked.

`Amount` sets the amount of agents, or landmarks, respectively.

Under `Type` the type of landmarks generated can be selected, from one of the following four options: `From topology`, `Grid`, `Random` (default) or `Random (clustered)`.

`Row obstacles` creates row-based obstacles based on the underlying grid when checked.

`Grid size` sets the grid size in local coordinates - a grid size of 1.0 would divide the environment in the middle in both directions.

`Grid rotation` sets the grid rotation in degrees.

Output settings

Under **Output settings**, the algorithm used to create a solution can be selected from the list of available algorithms.

If `Render output` is unchecked, the environment and output solution will not be rendered.

If all settings are as they should be, press `Create solution` to generate a solution to the loaded environment for the specified number of agents.