NTNU
Norwegian University of
Science and Technology

Handling Autonomous Robot Scheduling as an Optimization Problem

Alexander Bakke
Nils Inge Rugsveen

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

This dissertation investigates the possibility to model the behavior of an agent solving a complex robotic scheduling task, mission 7a of the International Aerial Robotics Competition, as an optimization problem known as the Time-Dependent Orienteering Problem with Time Windows. The robot environment is stochastic and dynamic, and the agent has to make decisions in real time, with little to no possibility for preprocessing. Solution techniques for the problem model and specializations of the problem have been investigated, along with the swarm algorithms Ant Colony Optimization Algorithm and Intelligent Water Drops Algorithm.

The swarm algorithms were implemented and applied to the problem model. The resulting system was used by a controller to solve the mission in a simulator, where the performance was evaluated by comparing the scheduler with a simple greedy controller.

Experiments show that the Intelligent Water Drops Algorithm and Ant Colony Optimization Algorithm were able to find solutions of adequate quality. Modeling the International Aerial Robotics Competition as a Time-Dependent Orienteering Problem with Time Windows showed great promise, and the scheduler performed better than the greedy controller.

The Ant Colony Optimization Algorithm had better performance than Intelligent Water Drops Algorithm in both solving the Time-Dependent Orienteering Problem with Time Windows and the mission. The Intelligent Water Drops Algorithm showed difficulties with completing plans when used in the controller because of large solution deviations. The performance of the system in the real world is uncertain, and partial observability may need to be addressed by Ascend NTNU before the competition.

Sammendrag

Denne avhandlingen utforsker mulighetene for å modellere oppførselen til en agent som løser en kompleks planleggingsoppgave med roboter, oppdrag 7a i konkurransen International Aerial Robotics Competition, som et optimaliseringsproblem kjent som det Tidsavhengige Orienteringsproblemet med Tidsvinduer. Robotmiljøet er stokastisk og dynamisk, og agenten må ta avgjørelser i sanntid, med liten til ingen mulighet for å gjøre forarbeid. Løsningsteknikker for problemmodellen og spesialiseringer av den har blitt utforsket, sammen med svermalgoritmene Maurkolonioptimalisering og Intelligente Vanndråper.

Svermalgoritmene har blitt implementert og anvendt på problemmodellen. Det utviklede systemet ble brukt av en kontroller for å løse oppdraget i en simulator, hvor ytelsen ble evaluert ved å sammenligne planleggeren med en enkel grådig kontroller.

Eksperimenter viser at Intelligente Vanndråper og Maurkolonioptimalisering klarte å finne løsninger av tilfredsstillende kvalitet. Å modellere International Aerial Robotics Competition som et Tidsavhengig Orienteringsproblem med Tidsvinduer viste lovende resultater, og planleggeren løste oppgaven bedre enn den grådige kontrolleren.

Maurkolonioptimaliseringen hadde bedre ytelse enn Intelligente Vanndråper både i å løse det Tidsavhengige Orienteringsproblemet med Tidsvinduer og oppdraget. Intelligente Vanndråper viste vanskeligheter med å fullføre planer når den ble brukt i kontrolleren som følge av store avvik mellom løsningene. Ytelsen til systemet i den virkelige verden er usikker, og delvis observerbarhet må kanskje bli adressert av Ascend NTNU før konkurransen.

Preface

This master dissertation was written by the authors at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU).

The project is a part of Ascend NTNU, which is a student driven organization at NTNU. Ascend NTNU will participate in the the International Aerial Robotics Competition(IARC), the longest running collegiate aerial robotics challenge in the world, in the summer of 2016 as the first Scandinavian participants. Ascend NTNU was founded during the spring of 2015 and consists of students from multiple disciplines ranging from first year to fifth year students.

The authors would like to thank supervisor Keith L. Downing for valuable guidance and insight during the research.

Also, Ascend NTNU and the members should be acknowledged for their good support and good memories, as well as KONGSBERG for the financial support.

Trondheim, June 7, 2016

Nils Inge Rugsveen, Alexander Seldal Bakke

Acronyms

ACO Ant Colony Optimization Algorithm

ACS Ant Colony System

ERS Exponential Rank Selection

FPS Fitness Proportionate Selection

GPS Global Positioning System

HUD Heuristic Undesirability Degree

IARC International Aerial Robotics Competition

IWD Intelligent Water Drop

IWDA Intelligent Water Drops Algorithm

LRS Linear Ranking Selection

NTNU Norwegian University of Science and Technology

OP Orienteering Problem

OPTW Orienteering Problem with Time Windows

POI Point Of Interest

RQ Research Question

TDO Time-Dependent Orienteering Problem

TDOPTW Time-Dependent Orienteering Problem with Time Windows

TSP Traveling Salesman Problem

VRP Vehicle Routing Problem

Contents

Chapter 1

Introduction

The International Aerial Robotics Competition (*IARC*) is the longest running collegiate aerial robotics challenge in the world, with the primary goal to further research in aerial robotics technology. Each time a mission is completed, a new one that has never been solved is issued. The current mission, mission 7a, has not been solved in the two tries so far.

The goal of mission 7a is to guide 10 robots driving on the floor of a square arena across a specified side using an autonomous aerial robot. The aerial robot can turn a ground robot a predefined angle by triggering a sensor on their top or front. The ground robots move in a repeatable pattern, but have some noise in their trajectory such that the exact trajectories and positions are uncertain.

Ascend NTNU was established by a group of students at the Norwegian University of Science and Technology (*NTNU*) to develop competence and knowledge in the area of robotics, aiming at participation in IARC 2016. This dissertation will contribute to Ascend NTNU and their participation in IARC with a system that controls the behavior of the aerial robot. Some of the challenges include action planning in a stochastic environment, moving targets, preventing ground robots from exiting the arena erroneously, and minimization of completion time.

The problem was abstracted and modeled as a Time-Dependent Orienteering Problem

with Time Windows (*TDOPTW*), a less known generalization of the Traveling Salesman Problem. The objective is to maximize the received reward for visiting a set of nodes in a graph, given a cost budget, i.e. maximum distance to travel, and a limited time window to visit each node. Additionally, each node has a service time the agent is required to wait before departing, and the model is time-dependent, meaning that the edge costs depend on the cost spent so far traversing the graph.

Due to rapid changes in the environment, finding an exact solution is assumed to be inessential, as such it was decided to use optimization algorithms for solving the problem. Furthermore, since these solutions have to be constructed quickly, it was decided to use the class of swarm optimization algorithms, which has shown to be computationally efficient at finding high quality solutions. Two algorithms were chosen for this purpose: Ant Colony Optimization Algorithm (*ACO*), which is inspired by how ants explore its environment, because of its strong foundation in research, many variants, and remarkable results on several problems closely related to TDOPTW; and Intelligent Water Drops Algorithm (*IWDA*), which is inspired by how water flows through its environment creating rivers to reach the ocean, because of its similarities to ACO though having a more complex model, and its convergence speed. To the extent of the authors' knowledge, neither of these algorithms have been shown to solve the Time-Dependent Orienteering Problem with Time Windows before.

1.1 Research Goal

The goal of this research is to implement a scheduling system, modeled as a Time-Dependent Orienteering Problem with Time Windows, for an autonomous aerial robot that guides ground robots in a dynamic and stochastic environment. The implemented system will be used in the 2016 International Aerial Robotics Competition, where continuous and time efficient evaluation of action plans is required in order to respond to changes in the environment.

1.2 Research Questions

Based on the competition environment and the proposed problem model, the following research questions have been formulated:

RQ1 Which of the algorithms Ant Colony Optimization and Intelligent Water Drops can solve the Time-Dependent Orienteering Problem with Time Windows?

RQ2 Which of the successful algorithms provides the best trade-off between solution quality and computation time?

RQ3 Is solving the IARC competition mission 7a as a Time-Dependent Orienteering Problem with Time Windows with the algorithms presented in RQ1 better than a greedy algorithm?

Research question 1 (*RQ1*) and 2 (*RQ2*) relates to the application of the algorithms on TDOPTW, while research question 3 (*RQ3*) relates to IARC mission 7a, with a great practical difference: In RQ3 the environment is defined as stochastic and continuous, which requires continuous repair of action plans and limits benefit of deliberation about future events. The results from RQ3 will show if modeling mission 7a as a combinatorial problem has any benefits compared with the simple approach of pursuing the highest available reward at each execution step.

1.3 Summary

The goal of this research is to implement a scheduling system for an aerial robot for Ascend NTNU, to use in the International Aerial Robotics Competition 2016, mission 7a. The problem environment has been modeled as a Time-Dependent Orienteering Problem with Time Windows, a generalization of the Traveling Salesman Problem. Intelligent Water Drops Algorithm and Ant Colony Optimization Algorithm, two swarm intelligence algorithms used in search, are proposed to solve the problem. The most appropriate algorithm,

including a simple greedy approach, to solve the mission will be identified.

In Chapter 2: Background the mission requirements, problem model, and swarm algorithms will be examined. Chapter 3: Tools and Technologies reviews the tools used and developed during the research, such as a simulator of the mission environment, while in Chapter 4: Methodology the implementation of the system will be explained and discussed. The results of the tests performed are shown and discussed in Chapter 5: Results and discussion.

Chapter 2

Background

In this chapter the International Aerial Robotics Competition and the mission will be described, and based on this the problem model defined. Problems related to the Time-Dependent Orienteering Problem with Time Windows, and their solution approaches, will be reviewed. A general explanation of the algorithms proposed to solve the problem will be given, as well as different variants and previous work found in the literature.

2.1 International Aerial Robotics Competition

The International Aerial Robotics Competition was started in 1991 on the campus of Georgia Institute of Technology, and is the longest running collegiate aerial robotics challenge in the world. The primary goal of the competition has been to provide a purpose to further research in state-of-the-art technology in aerial robotics, where each challenge required autonomous robotic behavior not demonstrated by any organization or government at the time. For a brief description of previous missions, see Appendix Appendix A.2.1: Previous Missions.

2.1.1 The Mission

The seventh and current mission was initiated in 2014, and consists of two parts designated as mission 7a and mission 7b. This dissertation will try to solve the planning problem of mission 7a. The overall goal is for the aerial robot to guide 10 autonomous ground robots across the correct side, designated as the 'green' side, of a square within 10 minutes. Although the task seems similar to a herding problem the robots does not act as a group, which is apparent by frequent collisions between robots and that they have no knowledge about the environment. Once a ground robot has moved outside the arena, whether over the green side or any other, it is out of the game. Only when mission 7a is completed will participants be tested in mission 7b.

In mission 7a, each team is allowed three flight attempts in order to complete the mission. The minimum mission, i.e. the minimum requirements for the mission to be deemed achieved, is to get at least 7 ground robots that have been landed upon to cross the green boundary.

2.1.2 Details

The following excerpt of conditions and rules are described by the organization IARC International Aerial Robotics Competition (2015).

2.1.2.1 Targets

There are 10 autonomous ground robots moving around the arena at the speed of 0.33 m/s, acting as targets for the aerial robot to interact with. The hobbyist robot iRobot Create, which can be found at www.irobot.com/About-iRobot/STEM/Create-2.aspx, serves as the basis for the robot hardware. The movement of the robots are decided by four factors:

- Every 5 seconds the trajectory is changed 0–20 degrees (the 'trajectory noise')

- Every 20 seconds the trajectory is changed 180 degrees (the 'trajectory reversal')
- On collision the trajectory is changed 180 degrees
- By landing on top of the robot, the trajectory is changed 45 degrees clockwise

Figure 2.1: IRobot Create is used as the ground robots. Image is taken from RobotShop Inc (2015), a robotics vendor located in Mirabel, Quebec, Canada.

That way the ground robots change trajectory deterministically every 20 seconds, and with a uniform probability distribution every 5 seconds. The possibility of collisions between ground robots or ground robots and obstacles increase the uncertainty in the environment significantly. Although initial positions and trajectories are known, the stochastic factors makes it unsuccessful to plan beforehand.

In addition to the ground robots, there are four obstacle robots present in the arena. The obstacles are based on the same robot model as the ground robots, but can be as much as 2 meters tall. The obstacles move clockwise concentrically with a radius of 5 meters and a speed of 0.33 m/s, and will not change trajectory during the run. If an obstacle collides with a ground robot (or possibly another obstacle), it will stop until it can move freely again.

2.1.2.2 Aerial Robot

The basic design of the aerial robot may be decided by the participant, as long as it complies with these requirements:

- Able to initialize launch and flight inside the arena autonomously
- Capable of sustained flight

- Equipped with an independently controlled termination mechanism
- Navigates without use of global-positioning navigational aids, such as GPS
- Carries all sensory equipment

Computations may be performed on a separate computer and information transmitted between the computer and the aerial robot.

2.1.3 Run Termination and Scoring

A run is terminated when either

1. There are no robots left in the arena
2. The time has run out
3. The aerial robot has violated any of the rules:
   (a) Collide with obstacle more than three times, whether in the air or on the ground
   (b) Fly more than 2 meters outside the arena boundaries
   (c) Fly outside the arena boundaries for more than 5 seconds
   (d) Reach altitude of more than 3 meters

Participants receive points in two categories: Effectiveness measures, and subjective measures. The metrics relevant for this dissertation are the reward received for successfully guiding a robot over the green boundary, and the per minute penalty until completing the mission. An overview of the scoring categories can be found in Appendix A.2.2: Scoring.

The team with the highest numerical score after the minimum mission is achieved will win the AUVSI Foundation Grand Prize and be declared winner of the mission 7a competition. In the event of a tie, the team with the fastest run is declared the winner. The score serves as a ranking of the participants, both in the event of success or failure to complete the mission.

2.2 Problem Definition

This section will study the technical aspects of the problem more closely, and define an abstraction of the problem such that it can be modeled mathematically.

2.2.1 Problem Abstraction

The scheduling problem of mission 7a can be considered a combinatorial optimization problem, where a set of nodes has to be interacted with before leaving the arena such that maximum reward is received. Travel between nodes and interactions take time, the position of all nodes will change with time, and hence also the associated cost of traveling between them. This makes the problem asymmetric, because the cost of traveling between two nodes depends on the sequence they are visited in. A problem that closely resembles the problem of mission 7a is the Time-Dependent Orienteering Problem with Time Windows (*TDOPTW*).

2.2.2 Time-Dependent Orienteering Problem with Time Windows

The TDOPTW is a generalization of the Traveling Salesman Problem (*TSP*), with multiple differences: In TSP the objective is to visit all nodes in a graph and return to the origin with minimal cost Applegate et al. (2007), while TDOPTW seeks to maximize the reward of visiting a subset of the graph within a given cost budget, where the travel cost between two nodes depends on the departure time, and rewards can only be obtained within a given time window Garcia et al. (2010). A time window designates the minimum and maximum amount of time that can be used to reach the node in order to receive the reward. The time spent must be derived from the cost spent and the speed of the agent, but in this dissertation the agent has a constant velocity of one unit, and thus the term ‘time

window' is interchangeable with 'cost window'. At each node the agent must wait a designated 'service time', which is added to the traveling cost.

To the best of the authors' knowledge, TDOPTW has only been proposed by Garcia et al. (2010), and under no other name than TDOPTW. No mathematical formulation for the problem has been defined, thus the one presented here is derived from the Time-Dependent Orienteering Problem (TDO) as defined by Fomin and Lingas (2002), and Orienteering Problem with Time Windows ($OPTW$) as defined by Kantor and Rosenwein (1992).

$s \in S$: A solution in the search space S

T_{max}: Maximum travel time

$t(j) = [t_a(j), t_b(j)] \mid 0 \leq t_a(j) \leq t_b(j),\ 0 \leq t_b(j) \leq T_{max}$: Time window for node j

$\{v(i,j) \mid i = 1, \ldots, n-1, j = 2, \ldots, n\}$: The set of edges given nodes i and j

$c(i,j,t) = (v(i,j), t) \mid t_a(j) \leq t + d(i,j,t) \leq t_b(j)$: A solution component, where v(i, j) is traversed with departure time t

$d(i,j,t)$: Time cost associated with adding solution component c(i, j, t) to the path

$b(j)$: Service time associated with adding solution component c(i, j, t) to the path

$r(j)$: Reward associated with node j

$s = \{c(1,j,0), \ldots, c(k,n,t)\}$: A solution as a set of solution components

The following additional constraints apply:

$c(i,j,t_1) \in s \mid c(k,j,t_2) \notin s$: No destination nodes should appear twice in the solution

$\sum_{c(i,j,t) \in s} d(i,j,t) \leq T_{max}$: Travel cost must not exceed cost budget

Given a objective function defined by

$$f(s) = \sum_{c(i,j,t) \in s} r(j) \tag{2.1}$$

the problem is to construct a path $s \in S$ from a specified origin, node *1*, to a specified des-

tination, node n, such that $f(s)$ is optimized. The reward of node j can only be collected during the interval $t(j)$, its specified time window, but the agent may wait at the node until the opening time. We assume that at most one time window is associated with each node.

2.2.2.1 Problems Not Covered by TDOPTW

Although the competition environment closely resembles TDOPTW, the following properties are not considered in TDOPTW:

- Partially observable: The environment is probably not fully observable, such that the aerial robot does not have perfect knowledge about the robots' position and trajectory
- Stochastic: Future states cannot be calculated deterministically due to the robots' trajectory noise
- Continuous: Due to the stochastic environment there are an infinite number of possible states and interaction points

After research question 1 and 2 have been solved, the stochastic and continuous properties of the environment must be handled in solving research question 3. The aerial robot's sensory capabilities at the point of the competition is not yet known, and thus partial observability will not be solved in this dissertation, although suggestions for solving it will be made in Chapter 6: Conclusion.

2.2.2.2 Previous Work

As far as the authors are aware, the Time-Dependent Orienteering Problem with Time Windows (*TDOPTW*) is not a well researched optimization problem, and only one article solving this optimization problem was found, under the name TDOPTW. Garcia et al. (2010) used the TDOPTW to model a Personalized Electronic Tourist Guide for mobile hand-held devices, where the goal was to create personalized routes that maximize the

tourists' satisfaction. They managed to generate good solutions in real time[1] in a real world environment, creating tourist routes for the city San Sebastian.

Model

Each destination, i.e node, was called a Point Of Interest (*POI*) and given a score which indicated how valuable the POI was. The POIs had opening hours which were modeled as time windows. A tourist could travel between POIs by either walking or using public transportation. The public transportation had different departures depending on the time of day, thus introducing time dependency.

Approach

They developed a hybrid approach which split the TDOPTW into two problems, TDO and OPTW and used a different heuristics on each of them. First they did an offline calculation of the average traveling time between two POIs, and stored them in a database, and solved the problem as a OPTW based on Vansteenwegen et al. (2009), which uses an Iterated Local Search.

After they obtained a solution for the OPTW with average travel times, they did a repairing procedure with real travel times between POIs, starting from the first POI in the route. If the real travel time was shorter than average they tried to move the visit towards the start, updating waiting time and departure time of the visit. If the real travel time was larger and made the route infeasible, they removed the POI from the route, moving other visits forward.

Results

The city of San Sebastian, located in the North of Spain, was used to generate 32 test instances. The problem model had around 50 POIs, and 26 public transport lines with 467 stops. The minimal time unit was 1 minute, and had four different cost budgets: 2, 4, 6,

[1] In Garcia et al. (2010), real time was defined as less than 5 seconds delay.

and 8 hours. They established that a tourist needed a cost budget of 20 hours to visit all the POIs, and observed that for an 8 hour cost budget they managed to collect around half of the maximum possible score. Their worst calculation time was less than 0.25 seconds.

2.2.3 Related Problems

Specializations of the TDOPTW, such as the Traveling Salesman Problem, have been researched extensively, and several successful approaches and techniques have been identified. The previous research on specializations will be investigated in order to gain insight in techniques and approaches to solve different constraints present in TDOPTW. Additionally, some other problems will be discussed in order to determine their relevance to the research questions.

2.2.3.1 Traveling Salesman Problem

The Traveling Salesman Problem (*TSP*) is a combinatorial problem where the objective is, given a set of cities and the travel cost between each pair of them, to find the cheapest path to visit every city exactly once and return to the origin city Applegate et al. (2007). The problem has been proven to be NP-hard Laporte (1991). The practical purposes of the TSP has given rise to several, more complex variations of the problem.

Common Solutions

Laporte (1991) has investigated common solution techniques for TSP. The problem can be solved by either using exact algorithms or approximation algorithms, depending on the size of the problem and the resources available.

Exact Algorithms

An exact algorithm is an algorithm that promises to find the optimal solution to a problem. In order to solve exactly, TSP is often represented as an Integer Linear Programming

problem, where the aim is to optimize a linear function, the objective function, which is subject to equality and inequality constraints that are also linear. In TSP the objective function is a measure of the cost of traveling a path, and the constraints ensures the validity of the path.

A commonly used exact solution technique used on Integer Linear Programming problems are the Branch-and-Bound algorithm, where the idea is to find optimal bounds for the solution. The problem is often formulated as a tree, and the algorithm systematically enumerates the candidate solutions, checking that the branch is inside the bounds before enumerating any further. The branch is discarded if it cannot find a better solution than the best found so far. Typically for TSP, an initial lower bound can be obtained by relaxing the constraints defined in the Integer Linear Programming problem.

Approximate Algorithms

An approximate algorithm does not necessarily find the optimal solution to a problem, but a satisfactory one according to some criteria. Such algorithms are often used where computation power is limited or low delay is essential, in order to find adequate solutions without traversing the complete search space. As mentioned, the TSP is a NP-hard problem, so it is not always feasible to obtain an exact solution. Much research has focused on developing heuristics to obtain an approximate solution. These approximations can be categorized as heuristics with a worst-case performance, or heuristics with good empirical performance. A simple way of approximating a lower bound for a symmetrical (undirected) TSP is to compute the length of the shortest spanning tree of the problem. This is shown to guarantee a lower bound within two times the optimal solution, and can be done in $O(n^2)$ time.

Some of the heuristics, categorized as either ‘tour construction procedures’ or ‘tour optimization procedures’, known to yield good empirical solutions are briefly explained below.

Tour Construction Procedures

In tour construction procedures one or more paths are generated in each iteration, by iteratively adding an edge to the path until the path is completed. Each edge is selected ac-

cording to some mechanism guided by the heuristic, until a satisfactory solution is found.

The nearest-neighbor algorithm adds the nearest neighbor of the current node to the path at each step. It is shown that the complexity is $O(n^2)$

Insertion algorithms start with a path of two nodes, and iteratively inserts a node chosen with respect to a given criterion into the path. Such a criterion can be the least cost or the node furthest away from the path. The complexity of this procedure varies between $O(n^2)$ and $O(nlogn)$ depending on the criterion being used.

The Patching algorithm is performed by constructing subtours (circuits not covering all nodes) in a TSP, and connecting the subtours together by the minimal cost until the solution contains only one circuit.

Tour Optimization Procedures

Instead of generating a tour as in tour construction procedures, tour optimization procedures modifies an existing path in order to improve the solution.

The r-opt algorithm considers a initial tour and removes r arcs from the tour, before tentatively reconnecting the remaining subtours in all possible ways. If any improvements are made, the new tour is set as the initial tour and a new round of r-opt is performed. This continues until no more improvements can be obtained. Generally an r-value of 2 or 3 is used. The 2-opt algorithm will be further discussed in Section 4.5: Local Search.

Simulated annealing is a successive method in combinatorial optimization, which is derived from material annealing. In order to bring a material to a minimal-energy solid state, it is heated until the particles are randomly distributed. Then it is gradually cooled down until it reaches a stable state. In combinatorial optimization the temperature variable T is initially high, which allows solutions in the neighborhood of another to be examined. Unlike the r-opt algorithm, worse solutions can be obtained, which reduces the probability of being trapped in a local optimum. As the temperature T decreases, fewer states are examined and the algorithm focuses in the area of the best found solution (i.e decreasing explo-

ration[2] in favor of exploitation[3]).

Tabu search also allows solutions to be deteriorated, but to divert the search from the vicinity of examined solutions, it manages a list of forbidden solution components in a 'tabu list'. Each time a new best solution is found, its components are added to the tabu list, where items persist for a given number of iterations. The Tabu search runs until a stopping condition is met, such as a predefined number of iterations or a solution with a high enough fitness[4].

Relevance

TSP is a well known problem that has been extensively researched. By modifying techniques devised for TSP with additional constraints, they may be applied to generalizations of TSP. The research in representing and solving TSP with swarm optimization algorithms is particularly interesting in context of the research questions, more so the algorithms that have not received as much attention on generalizations of the TSP. Variants of ACO and IWDA used on TSP are presented and discussed in Section 2.3: Algorithms.

However, dynamic cost and maximization of profits are not considered in TSP, nor visiting nodes in a certain time window. As such, several constraints have to be applied to TSP solutions to adapt them to TDOPTW.

2.2.3.2 Orienteering Problem

The Orienteering Problem, denoted by *OP*, is a generalization of the TSP. Its name originates from Tsiligirides (1984) and Chao et al. (1996). In OP, cities and traveling costs are defined as in TSP. However, a predefined origin and destination node is given, as well as a maximum allowed cost budget for the tour. Each city has a reward associated with it, which is obtained when the traveler visits the city. The goal of OP is to maximize the total reward collected constrained by the cost budget, and thus TSP is a special case of OP

[2]Investigate a diverse area of the search space
[3]Search in the vicinity of known, good solutions in the search space
[4]The quality of the solution

where the origin and destination is the same, all cities has the same reward, and the cost budget allows for all cities to be visited.

This problem is also known as the Selective Traveling Salesperson Problem, the Maximum Collection Problem, and the Bank Robber Problem Vansteenwegen et al. (2010).

Common Solutions

Since OP is a generalization of TSP, many of the methods mentioned previously can be extended to solve the OP. The survey Vansteenwegen et al. (2010) lists several solution approaches. As with the TSP, there exists both exact and approximate algorithms to solve the OP. Swarm optimization techniques have also proved successful on the OP in later years.

Relevance

Although OP is not as well researched as TSP, the extensions applied to common TSP solution techniques provide useful insight when applying the algorithms to TDOPTW. The application of the swarm-based optimization algorithm Ant Colony Optimization on OP is presented and discussed in Section 2.3.2.4: Metaheuristics for the Orienteering Problem.

The OP is closer related to TDOPTW than the TSP, because it incorporates the maximization of profits as its main goal, and does not require all nodes to be visited. Multiple constraints remain, though, to make it identical to TDOPTW.

2.2.3.3 Orienteering Problem with Time Windows

The Orienteering Problem with Time Windows ($OPTW$) is a generalization of the OP, with an additional constraint: Each node's reward can only be obtained within the node's time window (cost window), meaning that the cost used to reach the node must be larger than the opening time and smaller than the closing time to receive the reward. The OP is therefore a special case of the OPWT where each city has an opening time of 0 (alterna-

tively negative infinite) and a closing time of the cost budget (alternatively positive infinite) such that all rewards can be obtained at any time Kantor and Rosenwein (1992).

Common Solutions

The survey Vansteenwegen et al. (2010) lists some solution approaches to the OPTW, but the specific techniques have not great enough relevance to be reviewed here. The most important issue raised was that because of the time windows, tour improvement procedures would not always yield valid routes in OPTW. It was stated that there are no general and efficient solution to perform a local search, a category of tour improvement procedures, producing only valid solutions.

Relevance

Investigations show that local search procedures vastly improve the performance of Intelligent Water Drops Algorithm (*IWDA*) and Ant Colony Optimization Algorithm (*ACO*) when solving TSP, as discussed in Section 2.3: Algorithms, but the performance when solving OPTW is unknown. The problems with tour improvement procedures in OPTW should be investigated further, in case some work-around which improves the performance of IWDA and ACO can be found.

There is one constraint in TDOPTW that is not considered in OPTW, namely dynamic cost depending on departure time, which will be investigated in Section 4.1: Representing Time-Dependent Cost.

2.2.3.4 Other Problems

Mission 7a in IARC can be modeled as different types of problems. At first glance it may look like a typical herding (Strombom et al. (2014)) or pursuit-evasion problem (Hespanha et al.). However, the robots in the competition does not sense or model the environment and cannot act as a herd or evade any pursuers. Furthermore, their actions are limited to turning a set angle on contact, which makes them unable to align and cohere as in herds or

evade by proximity as in pursuit-evasion problems.

Another approach is to look at the problem as a classical planning problem, and the use of an automated planning approach such as Hierarchical Task Networks. Hierarchical Task Networks breaks down goals into smaller tasks incrementally, and finally tasks into primitive actions the agent can perform to alter the state of the environment Ghallab et al. (2004). The main objective of such a system is to find a plan that achieves high-level goals by low-level tasks, while solving dependencies between actions and their preconditions. Although it would be possible for such a system to create plans for the competition environment, given a good reward function or heuristic, it is impractical as there are no dependencies, and the single high-level task of each robot to direct them towards the green boundary is trivial to map to the two low-level actions available.

Vehicle Routing Problem (*VRP*) is another generalization of TSP where several agents cooperate to visit all nodes in a graph, with the goal of minimizing the cost Dantzig and Ramser (1959). Since there is only one agent in the IARC mission it is not directly relevant, but the problem has received much attention and research with ACO and IWDA which can benefit this research.

2.3 Algorithms

In the following sections swarm intelligence, combinatorial problems, and the two swarm intelligence algorithms 'Intelligent Water Drops Algorithm' (*IWDA*) and 'Ant Colony Optimization Algorithm' (*ACO*) will be presented. The previous applications of the two algorithms and their relevance to the Time-Dependent Orienteering Problem with Time Windows will also be reviewed.

2.3.1 Foundation

Swarm Intelligence

The term 'swarm intelligence' was first introduced in Beni and Wang (1989), and denotes a class of metaheuristic[5] algorithms that simulate a system of simple and social living beings, such as ants, termites, birds, and fish Parpinelli and Lopes (2011). Metaheuristic algorithms are approximate algorithms used to obtain results with reasonable quality in a reasonable amount of computation time for hard combinatorial and continuous optimization problems. In swarm intelligence algorithms, simple entities interact with each other and the environment according to a small set of rules, which as a system has an emergent property that cannot be achieved by any single entity alone. Swarm algorithms can be applied to a wide variety of problems including search, simulation, and prediction, depending on the specific algorithm, but require intimate knowledge of the problem to achieve the wanted emergent behavior.

The General Combinatorial Problem

This subsection describes the combinatorial problem as formulated in Dorigo and Blum (2005), and will be used to describe the algorithms in the following sections. A combinatorial optimization problem P is defined as follows:

$P = (S, O, f)$: The problem model

S: The search space

O: A set of constraints among the variables

f: An objective function $f \mid S \rightarrow \mathbb{R}^+$ to be optimized

where a solution $s \in S$ is defined as

$s = \{X(1), ..., X(n)\}$: A set of n solution components

[5] A technique using a heuristic that is independent of the problem

$X(i) = v(i,j) \in D(i) = \{d(i,1),...,d(i,|D(i)|)\}$: Domain values of $X(i)$

$c(i,j) = (X(i), v(i,j)) \in C$: A solution component in the set of solution components

$J(s_p)$: The set of feasible solution components for a given partial solution

The goal then is to find a globally optimal solution $s*$. As an example, if the objective is to minimize the cost of a solution, s* can be defined as such:

$$f(s*) \leq f(s) \ \forall s \in S \tag{2.2}$$

2.3.2 Ant Colony Optimization

The class of algorithms called Ant Colony Optimization (*ACO*) was introduced in 1991 with the algorithm Ant System Dorigo et al. (1991), as one of the first mainstream swarm intelligence algorithms Parpinelli and Lopes (2011). While no applications to TDOPTW has been found, ACO has been successfully applied to the closely related problem 'Team Orienteering Problem with Time Windows' Montemanni et al. (2011), reviewed in Section 2.3.2.4: Enhanced Ant Colony System for the Team Orienteering Problem with Time Windows.

The foraging behavior of real ants serves as inspiration for ACO, emulating the behavior of depositing pheromone as stigmergy[6]. When searching for food, ants will initially explore the surrounding areas until a food source is found, and deposit pheromone on its return trip from the food source. Both the same and different ants can distinguish this pheromone trail and its strength in order to navigate to previously found food.

[6] A mechanism of indirect communication between agents or actions.

2.3.2.1 Description

When the ant discovers a food source, the amount and quality of the food determines the amount of pheromone the ant deposits on each time step on its return trip. Other ants enhance the trail as well when returning with food. As a mechanism of negative feedback, i.e. suppressing the positive feedback of the pheromone to keep the system from stabilizing, the pheromone will evaporate over time. As such, longer trails will have a lower concentration of pheromone because ants need longer time to travel to the food source. A reasonably good food source nearby may be more beneficial to a colony than a higher quality food source further away, and the balance between positive and negative feedback makes each ant able to perceive which food source has the highest utility for the colony.

An element of probability when choosing an action is present in each ant, which enables exploration. Ants may ignore the strongest pheromone trail, or all pheromone trails altogether, to find better routes or food sources. As the survey Dorigo and Blum (2005) points out, this allows ants to find the shortest path between their nest and the best food source by the indirect communication of pheromone trails alone. Even if a path is obstructed, e.g. with a falling stick, the ants are able to quickly find the shortest path around it by exploring and establishing new pheromone trails.

2.3.2.2 Definition

In ACO the following extension to the combinatorial problem in Section 2.3.1: The General Combinatorial Problem applies:

$s(k) \mid k = 1, ..., m$: The solution of ant k

$J(s_p(k))$: The set of feasible solution components for the partial solution of ant k

$t(i, j) \in T$: Pheromone trail parameters associated with component v(i, j)

t_0: The initial pheromone level, which is a constant larger than 0

$f(s)$: Fitness function to be maximized

2.3.2.3 Pseudocode

The following pseudocode shows the general ACO framework, where the functions are described below.

```
1  Input: P, and parameters
2  InitializePheromoneValues(T)
3  s* = Null
4  while (not termination condition):
5   G = Ø
6   for(j = 1, ..., m):
7    s = ConstructSolution(T)
8    if(s is valid solution):
9     s = LocalSearch(s)
10    G = G + s
11    if(s* == Null or f(s) > f(s*)): s* = s
12  ApplyPheromoneUpdate(T, G, s*)
13 return s*
```

Figure 2.2: Pseudocode for the Ant Colony Optimization Algorithm

InitializePheromoneValues

Set the pheromone level of all edges to the initial pheromone level t_0.

ConstructSolution

Given

$o(i,j)$: The heuristic value of adding solution component c(i, j),

iteratively build a sequence of feasible solution components by, for each ant k, selecting a feasible solution component by the transition probabilities defined as

$$p_k(i,j) = \begin{cases} \frac{(t(i,j))^a \times (o(i,j))^b}{\sum_{c(q,r) \in J(s_p(k))} (t(q,r))^a \times (o(q,r))^b} & if\, c(i,j) \in J(s_p(k)) \\ 0 & else \end{cases} \quad (2.3)$$

where a and b denotes a relative importance between heuristic information and pheromone value. If the set of feasible solution components is empty, the function returns.

ApplyPheromoneUpdate

A global pheromone update function will be applied to all components as follows.

$$t(i,j) = (1-p) \times t(i,j) + \frac{p}{G_p} \sum_{\{s \in G_p \mid c(i,j) \in s\}} F(s) \tag{2.4}$$

where

$p \in (0,1]$: The evaporation rate of the pheromone

$G_p \subseteq G \cup s*$: Some subset of the newly found solutions and the best solution, depending on the problem

$F(s) \mid f(s) < f(s') \rightarrow \infty > F(s) \geq F(s')$: A quality function F(s)

LocalSearch

A local search is performed by applying local changes to a solution, creating new solutions that are closely related to the original solution. In the context of discrete combinatorial problems, a local change could be to swap the order of two solution components. Performing a local search is optional in ACO and not necessary for the algorithm to function properly, but may improve the performance drastically on larger problems. Local search provides exploitation in the neighborhood of found solutions to improve them, as a tour optimization procedure. Some of the most used and well-known local search algorithms to use with ACO are *r-opt*, as described in Section 2.2.3.1: Tour Optimization Procedures, and *Lin-Kernighan*, in which r and a variable number of components are exchanged respectively Dorigo and Gambardella (1997). In ACO, an r-value of 2 or 3 is usually applied, and 2-opt will be further explained in Section 4.5: Local Search.

2.3.2.4 Variants and Applications

Several variants of the ACO algorithm has been devised and successfully applied to various problems. Some of the variants are listed in Table 2.1. Table 2.2 includes some of the general applications for the ACO algorithm. Several authors have solved TSP implementing the general ACO as a tour construction algorithm, including Ant System in the original ACO article Dorigo et al. (1991). More interesting are the several variants of ACO that demonstrate improved performance on TSP and its generalizations compared to Ant System, and thus will receive the most attention in this section.

Algorithm	**Authors**	**Year**
Ant System	Dorigo et al.	1991
Elitist Ant System	Dorigo et al.	1992
Ant-Q	Gambardella and Dorigo	1995
Ant Colony System	Dorigo and Gambardella	1996
Max-Min Ant System	Stützle and Hoos	1996
Rank-Based Ant System	Bullnheimer et al.	1997
Ants	Maniezzo	1999
BWAS	Cordon et al.	2000
Hyper-Cube Ant System	Blum et al.	2001

Table 2.1: A non-exhaustive list of successful ACO algorithms Dorigo et al. (2006)

Problem name	Authors	Year
Traveling Salesman Problem	Dorigo et al.	1991, 1996
Graph Coloring	Costa and Hertz	1997
Vehicle Routing	Gambardella and Dorigo	1999
Multiple Knapsack	Leguizamón and Michalewicz	1999
Sequential Ordering	Gambardella and Dorigo	2000
Constraint Satisfaction	Solnon	2000, 2002
Project Scheduling	Merkle et al.	2002
Bayesian Networks	Campos et al.	2002
Orienteering Problem	Liang et al.	2002
Maximum Clique	Fenet and Solnon	2003
Team Orienteering Problem with Time Windows	Montemanni et al.	2011
Time-Dependent Orienteering Problem	Verbeeck et al.	2013

Table 2.2: A non-exhaustive list of ACO algorithm applications Dorigo et al. (2006)

Ant Colony System

A variant of ACO called Ant Colony System (*ACS*) has been shown to outperform Ant System and other nature-inspired algorithms such as simulated annealing and evolutionary computation as a tour construction algorithm on TSP Dorigo and Gambardella (1997).
The differences from Ant System are:

- The global pheromone update rule only applies to components in the best found solution
- A local pheromone update rule applies to edges when visited

- Modified state transition rule: If $u \leq u_0$ for a random float $0 <= u <= 1$ and some parameter u_0 as the exploitation probability, the edge with the highest transition probability is chosen; else the normal state transition rule is used

The local pheromone update rule is defined as

$$t(i,j) = (1-p) \times t(i,j) + p \times t_0 \tag{2.5}$$

One important characteristic of this modification is that the local pheromone update rule will instantly degrade the pheromone level of the traversed edge, such that other ants will be less inclined to pick these edges. Later when more edges are degraded the previously traversed edges will become relatively more attractive again and have a higher probability of being traversed late in a path.

The modification of the global pheromone update rule was done to improve exploitation of the most promising areas of the graph, although it is pointed out that the difference is minimal. The modified state transition rule leads to higher exploitation of edges with a large pheromone and heuristic value.

When compared with Simulated Annealing, Elastic Net[7], and Genetic Algorithm on relatively small problem sets, ACS generally had better solution quality and significantly lower computation time.

The algorithm was also extended to use a local search procedure, *restricted 3-opt*, which exchanges 3 components in the solution with unused components in order to find better solutions. Restricted 3-opt was chosen because of its performance on the asymmetric version of TSP, as it does not allow swapping to be a reversion of a path between two cities. As a result, it avoids parts of the search space that generally contains no improvements. Another criteria of restricted 3-opt is that it only swaps edges with better alternatives.

The test results showed that by extending the algorithm with a tour improvement heuristic, it was able to compete with the best algorithms of the previous year for solving large

[7] An iterative procedure where two forces are applied to a ring: One for minimizing the length of the ring, the other for minimizing the distance from the ring to each node

symmetric and asymmetric TSP, with significantly less effort.

Ant-Q

The Ant-Q family of algorithms are based on Ant System and inspired by Q-learning Gambardella and Dorigo (1995). It was shown to solve TSP with very good or optimal solutions on hard instances of the asymmetric problem version, where finding an exact solution is infeasible.

The state transition rule is the same as in ACS, while the equivalent to pheromone values, called AQ-value (reads "Ant-Q-value"), are updated for each city visited based on two terms: A reinforcement term and a discounted evaluation of the next state. The last term introduces a preference for paths connecting to short paths.

An interesting characteristic of this system is that the ants do not converge to a single path, but rather continues to optimize the results in a subset of the search space indefinitely.

Max-Min Ant System

The main goal of creating the Max-Min Ant System was to investigate the exploitation of the best results during a run Stültze and Hoos (1996). The main differences compared with Ant System are:

- Only the best ant adds pheromones
- Explicit maximum and minimum pheromone values
- Pheromone trails are initialized to the maximum value

It was observed that some configurations of the algorithm was prone to stagnate early, but could be fixed by adjusting the maximum and minimum pheromone values. It was also observed that the convergence rate could be adjusted by the pheromone evaporation rate.

Using a 3-opt local search procedure, it was shown to generate solutions with higher quality than the winner of the First International Contest on Evolutionary Computation on

asymmetric TSP, but had longer run times. It was pointed out that, although it is an improvement over Ant System, the results are suboptimal to other ACO variants, and the local search procedure could be improved by a large step Markov Chain, iterated Lin-Kernighan, or Genetic Local Search.

Metaheuristics for the Orienteering Problem

See Section 2.2.3.2: Orienteering Problem for a review of the Orienteering problem definition.

In order to comply with the cost restriction, the Ant System algorithm evaluates the cost of connecting a partial solution with the final node at each selection step. In Liang et al. (2002), if the cost budget is reached or exceeded, the ant will travel to the final node and terminate its run, as a result either binding to or breaching the constraint. Instead of refusing infeasible solutions and restricting the search space of the ants, a local search is performed to improve the solution and hopefully find a feasible solution.

A local search procedure called *Variable Neighborhood Search* is employed, where multiple local search procedures are employed sequentially, each using a larger neighborhood search space than the previous. Each procedure is initiated only if the preceding iteration could improve the solution. The search method is further explained in Section 4.5: Local Search.

In order to discourage ants from following trails with a cost that far exceeds the constraint, a penalty is incorporated in the pheromone update function, which size depends on the distance to the cost budget. The pheromone update is performed locally on each step, and globally on the best feasible solution. Generally, the local updates evaporate the trail, and the global updates enhance the trail.

Some comparisons with other algorithms have been made, and it showed similar or better computation times Liang et al. (2002).

A Fast Solution Method for the Time-Dependent Orienteering Problem

The Time-Dependent Orienteering Problem differs from the Orienteering Problem only in the costs, where each edge cost is dynamic and depends on the cost spent traversing the graph so far. The approach in C.Verbeeck et al. (2013), based on ACS, will be discussed in this section. The context of the article is creating a personalized tourist trip planner where travel time for a given distance depends on traffic congestion. Travel speed on an arc is derived from a speed model, based on congestion and the relevant time step.

While the solution for OP proposed in Section 2.3.2.4: Metaheuristics for the Orienteering Problem allows invalid solutions to be created before optimizing it to find a valid solution, this approach creates only valid solutions and use local search procedures to increase the number of nodes visited. *2-opt* is used after constructing a solution in order to improve the traveling cost, and then the *Insert Local Search Procedure* attempts to create valid solutions with more nodes by inserting non-included vertices at different indices.

The test results show that high quality solutions were achieved with low computational cost compared to other techniques. A sensitivity demonstration also showed that behavior did not change with small parameter changes, which indicates that robust behavior might be expected in real world applications. It is worth noting that the approach outlined here requires an effective tour optimization procedure, which as mentioned in Section 2.2.3.3: Orienteering Problem with Time Windows can be difficult on problems with time windows.

Enhanced Ant Colony System for the Team Orienteering Problem with Time Windows

The Team Orienteering Problem with Time Windows is mostly identical to the Orienteering Problem with Time Windows, the only difference being that multiple agents can cooperate to solve the task, which generally leads to higher received reward in total.

Montemanni et al. (2011) solved this problem using the Ant Colony System (described in Section 2.3.2.4: Ant Colony System) with two changes in the implementation to reduce the computational cost: First, in the state transition rule, instead of performing a fitness proportionate selection between all possible edges when $u > u_0$, it selects the edge with the corresponding position in the best solution found so far.

Second, a better integration between the constructive phase and the local search procedure, based on the assumption that repeatedly performing a local search will not lead to improvements, such that this procedure should be performed less frequently than every iteration for every solution. They propose to perform a local search with a probability of $\frac{i_c}{i_t}$, where i_c is the number of iterations since last local search on this solution, and i_t is the total number of iterations performed.

The benchmark results showed that when the algorithms were constrained to 3600 ms computation time on very large problem instances, their implementation generally performed better than the default implementation, because it was able to converge to a good solution faster. With no time constraint it was able to improve the best known result on some larger problem instances with two to four agents in the team, but no improvements were reported using one agent (equivalent to the Orienteering Problem with Time Windows). Therefore these improvements are only expected to be of value if high computational cost becomes a problem.

Other

Investigations on the Dynamic Traveling Salesman Problem showed that using an immigrant scheme was beneficial in dynamic environments Mavrovouniotis and Yang (2013). In the Dynamic Traveling Salesman Problem, some property that affects the cost of an edge is changed, in a stochastic fashion, during the path traversal. In order for the agent to always travel the optimal route, according to the available knowledge, it has to optimize the problem several times. An immigrant scheme works by retaining the pheromone trails between each optimization iteration, but replacing the least fit part of the established population with new individuals, in order to both maintain diversity and provide knowledge transfer

between iterations. This technique is relevant when similar problem states needs to be calculated consecutively and consistency or computational cost are an issue.

Yu et al. (2009) created the *Improved Ant Colony Optimization* algorithm for solving the Vehicle Routing Problem. For a description of the problem see Section 2.2.3.4: Other Problems. They introduced a mutation operation with was executed with decreasing probability for each iteration, in which two solution tours are mixed to create two new solutions, and an ant-weight strategy, where local pheromone updates are based on the contribution of each edge to the solution. The algorithm had a little higher run time, but generated better results compared with some of the best algorithms on the problem.

2.3.3 Intelligent Water Drops

Intelligent Water Drops Algorithm (*IWDA*) is a nature-inspired swarm optimization technique introduced in 2007 by Shah-Hosseini (2007). No applications to neither TDOPTW nor OPTW have been found. The algorithm is inspired by how water drops flowing in rivers find their way to lakes, seas or oceans despite obstacles in their way. If there were no obstacles the water drops would flow straight towards the lowest point due to gravity, which is the shortest path. However, due to the natural obstacles in the environment, the river has twists and turns. It seems that nature creates optimal paths in terms of distance from the destination and the constraints of the environment Alijla et al. (2014).

2.3.3.1 Description

Imagine a water drop moving from one point in a river to another as show in Figure 2.3. It is assumed that the water drop has the capacity to carry soil and as the water drop moves to its destination the soil carried is increased as the soil in the river bed decreases.

Figure 2.3: The IWD on the left flows to the right while removing soil from the river bed and adding it to itself

As well as carrying soil, the water drop has a velocity which plays an important role in removing soil from river beds. If two water drops with the same amount of soil traverse the same river stretch it is assumed that the water drop with higher velocity removes more soil from the river bed. In Figure 2.4 higher velocity is represented by a longer vector and the size of the water drop represents the soil carried by the water drop.

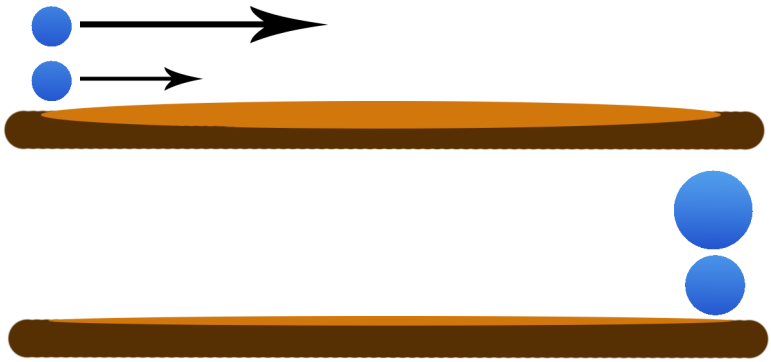

Figure 2.4: The IWD with higher velocity retrieves more soil from the river bed

In contrast to how the velocity of water drops traversing a path determines the amount of soil removed from the path, the amount of soil in the path determines the velocity of a water drop. Figure 2.5 shows two identical water drops traversing two different paths. The path with less soil lets the water drop gain more speed and gather more soil from the path, while the path with more soil provides greater resistance, hence lower velocity for the drop and less soil gathered.

Figure 2.5: Two identical IWDs flows in two different rivers. The IWD that flows in the river with less soil gains more speed and gathers more soil

Water flowing in nature chooses the path of least resistance, which for the Intelligent Water Drop (*IWD*) is the soil in the edges to traverse, thus an IWD has a higher possibility to choose an edge with less soil over an edge with more soil.

In nature, countless of water drops flow together and changes the environment to find an optimal path for reaching their destination. IWDA does the same with agents constructing paths and changing the search space over time, using the characteristics mention above, until the optimal or near optimal path emerges.

2.3.3.2 Definition

Extending the combinatorial problem defined in Section 2.3.1: The General Combinatorial Problem, the IWDA can be defined as follows, based on the definition in Alijla et al. (2014):

$s(k) \mid k = 1, ..., m$: The solution of water drop k

s_{IB}: Iteration best solution

vel^{IWD}: The velocity of a given IWD

$soil(k)$: Soil carried by a water drop k

$soil(i, j)$: Soil retained by a component c(i,j)

2.3.3.3 Pseudocode

The main steps of the IWDA are shown in Figure 2.6, and its details are defined below. Only the most important equations are described here, see Appendix A.1: Complete Formulation of Intelligent Water Drops Algorithm Equations for all of the equations that make up the Intelligent Water Drops algorithm.

```
1  Input: P, and parameters
2  InitializeEdgeSoil()
3  while(not algorithm termination condition):
4    InitializeWaterDrops()
5    while(not construction termination condition):
6      for (k = 1, ..., m):
7        EdgeSelection()
8        UpdateWaterDrop()
9        if (f(s(k)) < f(s_IB)):
10         s_IB = s(k)
11   UpdateEdges(s_IB)
12   if (f(s_IB) < f(s*)):
13     s* = s_IB
14 return s*
```

Figure 2.6: Pseudocode for the Intelligent Water Drops Algorithm

InitializeEdgeSoil

Initialize each edge in the graph with a set amount of soil.

InitalizeWaterDrops

Creates the IWDs and distributes them randomly over all possible nodes.

EdgeSelection

As mention earlier, an IWD has a higher probability of selecting a path with less resistance, i.e. less soil. It uses a fitness proportionate selection, described in Section 4.3: Se-

lection Methods, where the fitness is measured as the amount of soil residing in an edge:

$$f(soil(i,j)) = \frac{1}{\epsilon + g(soil(i,j))} \quad [8] \tag{2.6}$$

$$g(soil(i,j)) = \begin{cases} soil(i,j) & if \min_{l \notin vc(IWD)} soil(i,l) \geq 0 \quad [9] \\ soil(i,j) - \min_{l \notin vc(IWD)} soil(i,l) & else \end{cases} \tag{2.7}$$

where *g(soil(i, j))* is used to shift the value of *soil(i, j)* towards positive values. Therefore IWDA represents the best path with a minimal fitness value, as shown in Figure 2.6, line 12.

UpdateWaterDrop

Here the velocity of and soil carried by the IWD, and soil residing in the traversed edge is updated. The most important equation for soil update is Equation 2.8 which describes the time an IWD uses to traverse an edge. The HUD(i, j), i.e. Heuristic Undesirability Degree, is a method that gives an indication of how poor the given move is. Thus the HUD should yield a low value if moving to j is a favorable move. HUD is discussed more in Section 4.2: Heuristic Functions. The faster an IWD traverses an edge, the more soil is transferred from the edge to the IWD.

$$time(i,j,vel^{IWD}) = \frac{HUD(i,j)}{vel^{IWD}} \tag{2.8}$$

[8] ϵ is a small positive number, usually 0.01
[9] vc(IWD) is a list of visited nodes for the IWD

UpdateEdges

After all IWDs in an iteration has completed constructing a path, the best IWD is chosen and all the edges in its path get updated according to this equation:

$$soil(i,j) = (1+\rho_{IWD}) \times soil(i,j) - \rho_{IWD} \times soil(s_{IB}) \times \frac{1}{(m-1)} \tag{2.9}$$

where m is the number of IWDs and ρ_{IWD} is a positive constant between 0 and 1.

2.3.3.4 Variants and Applications

This sections reviews the application of the algorithm on the Traveling Salesman Problem and the Vehicle Routing Problem, described in Section 2.2.3.1: Traveling Salesman Problem and Section 2.2.3.4: Other Problems respectively, as well as a proposed modification of the algorithm, and its convergence properties.

Solving the Traveling Salesman Problem

When the IWDA was first proposed, it was tested on typical TSP problem instances such as eli51 Reinelt (2008). Shah-Hosseini (2007) showed that the IWDA were able to find *global optimum* tours, however this was not guaranteed. Sometimes the IWDA converged to a good *local optimum* after few iterations. They also noticed that the IWDA were able to escape some local optima, which is an appealing property of the IWDA.

Solving the Vehicle Routing Problem

Kamkar et al. (2010) used the IWDA to solve the Vehicle Routing Problem described in Section 2.2.3.4: Other Problems. They compared it with implementations of Simulated Annealing and Tabu Search by Osman (1993), and Improved Ant Colony by Bin et al. (2008), on 14 benchmark instances designed by Christofides et al.

Their results shows that IWDA competes with other well known and widely used algo-

rithms for the problem. As discovered by Shah-Hosseini (2007), IWDA has the ability to escape local optima. IWDA was also shown to use consistently low computation time compared to the other algorithms.

Modification of Selection Method

Alijla et al. (2014) proposed two new *selection methods* to the IWDA (a description of 'selection methods' can be found in Section 4.3: Selection Methods). They pointed out that the original selection method, *fitness proportionate selection* (*FPS*), shown in Equation 4.3.1, were susceptible to three limitations:

1. Inability to accommodate negative soil values
2. Inability to create a different selection pressure for fitter nodes when most of the nodes have a similar fitness value
3. Inability to handle selection dominated by a node with an exceptionally high fitness value, as compared with those from other nodes in a selection pool

To address these limitations they proposed to use two types of ranking selection: Linear Ranking Selection (*LRS*), and Exponential Ranking Selection (*ERS*). While FPS is based on the absolute fitness value, ranking selection is based on the rationale of fitness rank to determine the probability of selection. The ranking of the fitness values are subject to the mapping function, which maps a fitness value to a selection probability, and the *selection pressure* (*SP*), which controls the degree of exploration and exploitation.

$$P(i) = \frac{1}{N} \times (SP - 2(SP-1) \times \frac{i-1}{N-1}) \tag{2.10}$$

$$P(i) = SP^{i-1} \times \frac{1-SP}{1-SP^N} \tag{2.11}$$

LRS is shown in Equation 2.10 and ERS in Equation 2.11. In both equations $i \in \{1, ..., N\}$ represents the rank of the edge, where 1 is the fittest edge (i.e. the edge with the lowest

soil value) and N is the least fit edge. In Equation 2.10 SP, where $1 \leq SP \leq 2$, is used to control the gradient of the linear selection function. In Equation 2.11 SP, where $0 < SP < 1$, is used to control the gradient for the exponential selection function.

Alijla et al. (2014) did an extensive study on different optimization problems, where TSP was one of them. Their study showed that the selection method had a significant impact on the performance of IWDA, and that ERS were more effective than LRS and FPS in respect to solution quality. However, the computational effort in using ranking selection was high, and the effort of ERS was shown to be 1.15 to 13 times higher than FPS in their case studies.

Convergence Properties of Intelligent Water Drops Algorithm

Shah-Hosseini (2008) investigated the convergence properties of the IWDA, and showed mathematically that the IWDA possess the property called 'convergence in value', which means that the algorithm is able to find the optimal solution if the number of iterations is sufficiently big.

2.4 Summary

The International Aerial Robotics Competition was initiated in order to further research in the field of autonomous aerial robotics behavior. Mission 7a was first tried in 2014, and has not been completed in the two tries so far. The goal of the mission is to autonomously guide 10 robots moving on the floor across a specified side of an arena using an aerial robot. The trajectory of the ground robots are changed every 5 seconds to induce stochasticity in the environment, and reversed every 20 seconds. In order to guide the robots the aerial robot can either land on top of them or induce a collision to make them turn a predefined number of degrees. As part of the challenge, the aerial robot must carry all sensory equipment and navigate without global-positioning system.

In order to solve the problem, the different mission requirements and the environment was

examined to determine how the aerial robot should behave. When considering a limited number of interactions, the problem can be considered a combinatorial problem of possible combinations of visitation order. The currently existing problem model with the highest fit with the mission abstraction was found to be the Time-Dependent Orienteering Problem with Time-Windows: Given a set of nodes where each has a position, trajectory, speed, reward, time window, and service time, find the optimal path (or visitation order) with respect to received reward, given that travel cost between two nodes change with time and the reward can only be collected within the node's time window. The environmental properties of the mission not handled by the TDOPTW problem model are the stochasticity, continuity, and partial observability, where the last property will not be solved in this dissertation.

Research showed that TDOPTW has only been examined once before, which solved the problem as an Orienteering Problem with Time Windows offline using the average cost of each edge over time, and repaired the solution with real-time data cost data to achieve TDOPTW. One difference in implementation is that it modeled the time-dependent cost as a series of incidents, opposed to the changing Euclidean distances most fitting the mission problem. Related problems include TSP, OP, and OPTW, where the main difference between each of them and between them and TDOPTW are additional constraints. As such, techniques used in specializations of TDOPTW is expected to work for TDOPTW as well.

Swarm algorithms belong to the field of Swarm Intelligence, where the general idea is to simulate a system of simple beings where an emergent property is achieved through social interactions. Ant Colony Optimization Algorithm and Intelligent Water Drops Algorithm are two such algorithms, used in search. ACO is inspired by the foraging behavior of ants, where emergent behavior is achieved by ants depositing pheromone trails, distinguishable by it and other ants, on paths leading towards food sources. Better food sources have stronger pheromone trails leading towards them, and equivalently in the algorithm for better solutions. An element of probability over which path to choose exists both in the algorithm and real world, which enables exploration of the search space. Variants of ACO include the commonly used Ant Colony System, and different applications of local search

procedures such as 2-opt. IWDA is inspired by how water flows and creates rivers in nature, where emergent behavior is achieved by water drops removing soil from the rivers. Rivers with less soil leads towards better solutions, and are more probable to be chosen by a water drop than a river with much soil. Variants of IWDA use different selection methods, which yield slightly better results with significantly more computational cost.

Chapter 3

Tools and Technologies

In the following sections the technologies used and the tools implemented in conjunction with the research will be described, and the motivation and importance of the tools will be briefly discussed.

3.1 Languages and Frameworks

The entire code base for this dissertation was implemented in C++11 by the authors. The project was built with GNU Make, which made it easy to build all the modules of the project.

The graphical modules, such as the simulator and visualization tool, are realized by Simple DirectMedia Layer 2 (SDL2). SDL2 is a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware through OpenGL and Direct3D.

3.2 Visualization Tool

When working with swarm based algorithms and optimization problems, simply implementing the algorithm is not enough. Each algorithm may have different parameters that

affect the behavior of the algorithm, such as number of iterations, number of agents and selection methods. There is rarely one setting that works for every problem, therefore these parameters needs to be experimented with. The parameters and selection methods will be discussed further in Chapter 4: Methodology. To quickly alter the parameters and the algorithms for the same problems, the authors developed a visualization tool.

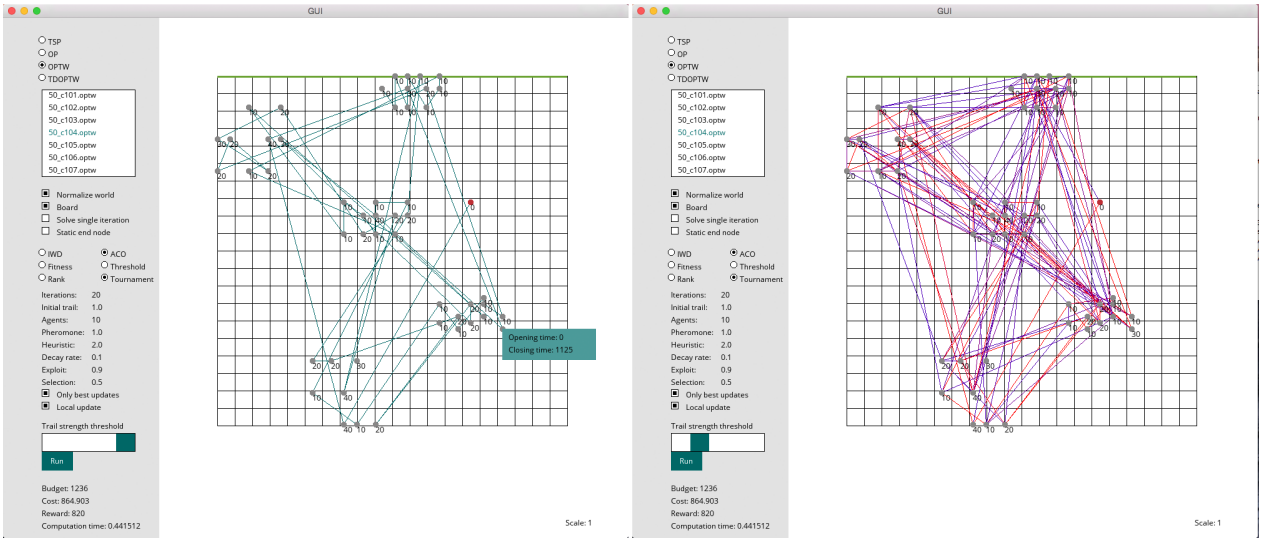

Figure 3.1: Two screen captures of the visualization tool, both showing an OPTW problem solved by ACO. To the left the solution is shown. To the right the pheromone trails are shown, where red symbolizes a high amount and blue a low amount of pheromone

3.2.1 Description

The visualization tool can read TSP, OP, OPTW, and TDOPTW problems from text files and render them as a graph. It can run both ACO and IWDA with different parameters and draws a teal line to show the resulting path of the algorithm. Figure 3.1 shows a screenshot of the visualization tool. To the right it shows the pheromones of the ACO. Red indicates a strong pheromone trail and blue indicates weak pheromone trails. For the IWDA this represents the amount of soil that resides in the edges. Underneath each node the node's reward is shown, and the time window and service time of a node can be seen by hovering over it with the mouse.

3.3 Simulator

A simulator was implemented in order to have a predictable testing environment, as well as being able to test independently of hardware. At the time this dissertation was written, the aerial robot was not available for testing. The mission simulator provided a testing environment in accordance with the mission environment, to determine the effectiveness of solving the mission as a TDOPTW.

3.3.1 Description

The simulator is a 2D representation of the mission environment and is shown in Figure 3.2. The yellow circles represents the ground robots and the blue circles represent the obstacle robots. The implementation of the robots is retrieved from the source code to be implemented in the physical ground robots, issued by IARC. The aerial robot is represented by the the white cross, the small white circle represents the destination of the aerial robot and the big circle is the perception area of the aerial robot. The perception area is where the aerial robot can be certain that it gets correct visual input used for landing, but is a feature not used in this dissertation as the whole environment is fully observable during this research. The white lines in Figure 3.2 represent the solution path of the algorithm and the red lines shows the calculated flight path for the aerial robot.

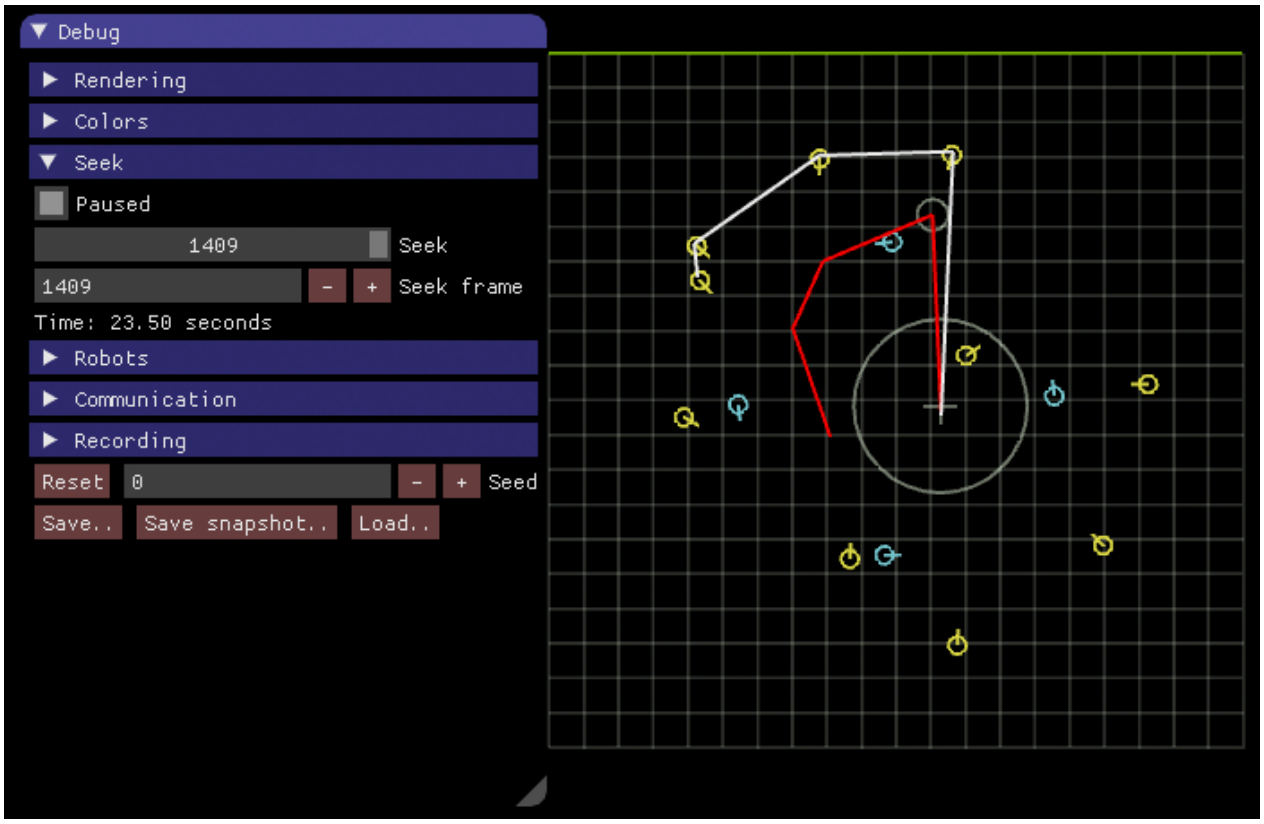

Figure 3.2: A screenshot of the simulator. Yellow circles are ground robots, blue are obstacles, the aerial robot is represented as the white cross, the small white circle is the destination of the aerial robot. The white path shows the robots to be visited and the red path shows the aerial robot's flight plan

The panel to the left shows the options of the simulator. The most important features of the panel are as follows:

- Pause the simulation
- Reset the simulation
- Record the simulation
- Generate problem instances from the simulator by pressing 'Save snapshot'. This saves the current state of the simulator which can be translated to a TDOPTW problem.

3.3.2 Controlling the Aerial Robot

An UDP (User Datagram Protocol) port is opened by the simulator. This port is used by the simulator to send and receive data. At a given interval the simulator is sending the state of the current simulation out on the UDP port. The state consist of the position and direction of all objects in the simulator, which robots are turning and the speed of the aerial robot. The simulator is also listening for commands at the UDP port. This makes the simulator independent of any specific controller, and makes it easy to implement additional controllers. The supported commands is listed below:

- LandOnTopOf(robot_ID)
- LandInFrontOf(robot_ID)
- Track(robot_ID)
- Search(x, y), where (x, y) is the destination coordinates
- Pause
- Start

Using these commands, another program can control the aerial robot by connecting to the same UDP port. A typical controller will listen to the port and receive information about the simulation. Based on this information it can decide what to do and send commands to the aerial robot. The implementation of the controller using TDOPTW is explained in Section 4.8: Applying the Model to the Simulator.

3.4 Summary

The code for this project was written in C++, using the library 'Simple DirectMedia Layer 2' for visualizations. In addition to implementing the algorithms a tool for visualizing different properties of the algorithms and their outputs, as well as a mission environment simulator and a controller for the aerial robot in the simulator, were created. The visualiza-

tion tool provides a user interface for changing parameters and loading different problem instances, and visualizing the algorithms' trails and outputted solutions in order to detect deficiencies. The mission simulator provides a testing environment in accordance with the mission environment, to determine the effectiveness of solving the mission as a TDOPTW.

Chapter 4

Methodology

In this chapter implementation details, the application of time-dependent cost, and modeling the mission environment as a Time-Dependent Orienteering Problem with Time Windows will be reviewed. Lastly the greedy controller will be described.

4.1 Representing Time-Dependent Cost

As the edge costs in TDOPTW are changing with time (imagine the nodes moving around), simply calculating the Euclidean distance between the initial position of two points is not sufficient. In Garcia et al. (2010) the time-dependence of the traveling cost is represented as a matrix, where each row represents a directed edge in the graph, and each column the travel cost after visiting the column index number of nodes so far. This is practical for their application, as it is possible to capture complex non-linear cost relationships in a straightforward and understandable manner.

In the case of the competition, on the other hand, it would be hard to model cost this way, as there are usually just one node that can be precisely reached with a given cost, and there are a lot of different combinations of paths, making the matrix impractically large. Furthermore, assuming linear movement between each trajectory reversal the cost can easily be calculated by finding the lowest intersection time between the robot and aerial

robot, as described below in Section 4.1: Intersection Time, and multiplying the time by the aerial robot's speed.

One aspect to remember when calculating edge cost is that all nodes move when the agent is visiting other nodes, so when calculating edge costs the position of the destination node must be calculated with respect to the cost used so far.

Two different approaches have been considered: Modeling the edges in OPTW with a dynamic cost function, or expanding the search graph with each possible edge cost. Given that the mission environment requires fast solution creation in order to keep up with the stochastic environment, it was chosen to use a dynamic cost function, as the expanded graph would be magnitudes larger. The changing cost and reuse of edges between different solutions could be a problem for ACO and IWDA, as the trail of each edge is affected by the solution combinations previously explored. While ACO might mitigate this problem as pheromones are degraded when better subspaces and solutions are found, IWDA can be severely biased by previously explored neighborhoods when finding better neighborhoods later.

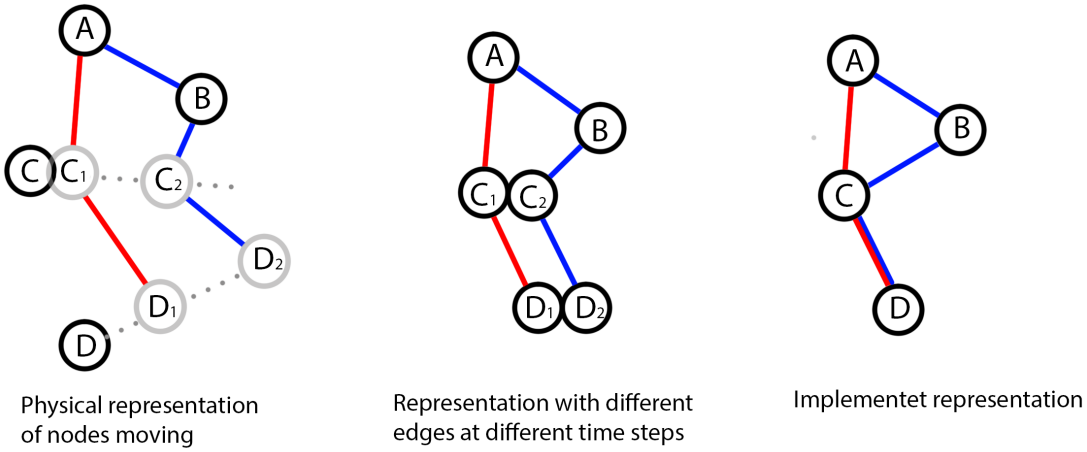

Figure 4.1: The left graph represents a generic representation of moving nodes, the other graphs two different ways to represent time-dependency. Node A and B are stationary, whereas C and D has a movement represented by the gray doted line. The red and blue lines represents two different paths.

Intersection Time

Given

U: The aerial robot's absolute speed

V: The robot's speed vector

R: The aerial robot's position vector

S: The robot's position vector

we want to find the time where the positions of the robot and the aerial robot are equal

$$\begin{aligned} S + Vt &= R + Ut \\ R - S + Vt &= Ut \\ P + Vt &= Ut \end{aligned} \tag{4.1}$$

The positions can be expressed as the distance from the origin of the system

$$\begin{aligned} (P_x + V_x t)^2 + (P_y + V_y t)^2 &= (Ut)^2 \\ P_x^2 + P_y^2 + V_x^2 t^2 + V_y^2 t^2 + 2P_x V_x t + 2P_y V_y t - U^2 t^2 &= 0 \\ (V_x^2 + V_y^2 - U^2) t^2 + (2P_x V_x + 2P_y V_y) t + (P_x^2 + P_y^2) &= 0 \end{aligned} \tag{4.2}$$

Then we can solve for t by using the quadratic equation.

4.2 Heuristic Functions

Heuristic functions are used to find approximate solutions to problems, trading accuracy for speed. ACO and IWDA use heuristic functions to select the next edge in the path when searching the graph. The implemented heuristic functions are described below.

Intelligent Water Drops Algorithm

IWDA uses heuristics for calculating the transition probability to each eligible edge. As explained in Section 2.3.3.3: EdgeSelection, the probability of traversing an edge is based on the amount of soil residing in an edge which again is dependent on Equation 2.8. This equation is dependent on the velocity of the drop and the Heuristic Undesirable Degree (HUD).

HUD must be adjusted for the given type of problem. As the goal of TDOPTW is to maximize the total received reward with a traveling budget, this HUD will disfavor long distances with low reward. HUD represents a favorable transition as a low value. Given the reward $r(j)$ received for adding edge $v(i,j)$ with the cost of $d(i,j,t)$ to the solution at time t, the heuristic function is implemented as

$$o(i,j,t) = \frac{d(i,j,t)}{r(j)} \tag{4.3}$$

Ant Colony Optimization Algorithm

In ACO the heuristic function is applied when calculating the transition probability to each eligible edge, as described in Section 2.3.2.3: ConstructSolution, in combination with the pheromone value. The pheromone value and heuristic value has their separate static parameters scaling the values, so that the relative importance may be adjusted.

As the goal of TDOPTW is to maximize the total received reward, this heuristic function favors edges leading to high rewards. Given the reward $r(j)$ received for adding edge $v(i,j)$ to the solution at time t, the heuristic function is implemented as

$$o(i,j,t) = \frac{r(j) - r_{min}}{r_{max} - r_{min}} \tag{4.4}$$

where

$$r_{max} = \max_i \; r(i)$$
$$r_{min} = \min_i \; r(i) \tag{4.5}$$

4.3 Selection Methods

Selection methods, mostly used in genetic algorithms, are genetic operators for selecting the best individuals from a larger pool of individuals, with some degree of diversity among its population Floreano and Mattiussi (2008). In ACO and IWDA selection methods are used when selecting the next edge, each having a fitness value based on some heuristic, to be added to the path of a solution. The reason for wanting diversity is because higher diversity is able to both capture the best individuals and a broad selection with opportunities to find better optima, opposed to always selecting the perceived best individuals which might lead the population towards a local optimum. The 'selection pressure' adjusts the competition among the individuals, and a higher selection pressure usually leads to lower diversity and faster convergence of the algorithm.

Rank selection was not tested on IWDA because Alijla et al. (2014) showed that it provided only minor improvements in result over fitness proportionate selection, with a large computational penalty. According to Floreano and Mattiussi (2008) tournament selection provides a middle-ground between rank selection and fitness proportionate selection in terms of effect, so in context of the knowledge about rank selection it was decided to not test tournament selection either on IWDA. All three methods were implemented and tested with ACO.

4.3.1 Fitness Proportionate

In fitness proportionate selection, also known as roulette wheel selection, an individual is selected according to a probability proportionate to the ratio between its fitness value and

the sum of the fitness values of the population. The probability is given by

$$p(i) = \frac{f(i)}{\sum_{j=1}^{N} f(j)} \tag{4.6}$$

where *f(i)* is the fitness value of individual i in a population of N individuals.

This can be illustrated by a roulette wheel, where each slots corresponds to an individual, and the size of the slot is proportionate to the fitness of the individual: The higher fitness an individual has, the higher the probability that it will be selected when the wheel is spun. Initially when all individuals have equal fitness, they have equal probability of being selected. In later iterations individuals with higher probability will dominate the selection process, which will sometimes cause premature convergence, meaning that it gets stuck in a local optimum.

4.3.2 Rank selection

Instead of selecting an individual with a probability proportionate to their absolute fitness level, in rank selection the individuals are ranked from best to worst based on the fitness level. The probability of selecting an individual is then calculated to be proportionate to their rank, where the best ranked individual has the highest probability of being selected. Given that each individual i has a rank $p(i) = i$ where *p(1)* is the least fit individual and *p(N)* is the fittest individual, the selection probabilities can be expressed similarly to that of fitness proportionate selection:

$$p(i) = \frac{p(i)}{\sum_{j=1}^{N} p(j)} \tag{4.7}$$

in a population of N individuals.

This alleviates the problem with a dominating set of high fitness individuals in fitness proportionate selection, but is more expensive computation-wise which can have a certain im-

pact when delay is crucial.

4.3.3 Tournament selection

In tournament selection multiple competitions between a subset of the population are performed, where only the best individual is selected in each competition. This procedure is performed n times in order to select n number of individuals. Each competition is performed by randomly picking k number of individuals, called the tournament size, from the population and selecting the fittest individual, i.e. selecting i such that

$$\max_{i \in k} f(i) \tag{4.8}$$

After each competition all the k individuals are put back in the population and are eligible to participate in later tournaments. Tournament selection provides a good compromise between selection pressure and diversity in the resulting population.

4.4 Pruning the Search Space

TDOPTW has several constraints that can be leveraged in order to prune the search space during the search. The constraints, as defined in Section 2.2.2: Time-Dependent Orienteering Problem with Time Windows, is repeated here for convenience:

- Each node can be visited only once
- Path cost cannot exceed cost budget
    - If an end node is specified, the edge connecting the destination node to the end node needs to be considered as well

Additionally, when the received reward is zero, whether because the node has an inherent reward of zero or because of missing the time window, it should be avoided as visiting it

will only result in increased cost.

In each path incrementation the nodes that yield reward without invalidating the path are extracted from the search space. This way the initially large search space may be reduced dynamically to exclude solution configurations that definitely would disadvantage the solution. Depending on the problem configuration, the set of viable nodes may be reduced to just a single or a handful of nodes.

4.5 Local Search

A local search is a tour improvement procedure, aiming at making small changes to a given solution in order to improve it. In this dissertation, the methods 2-opt and Variable Neighborhood Search have been tested on ACO.

2-opt

This simple local search mechanism works by taking any subset of the path (except the start and end node if they are constrained to specific values), and reversing the order of this subset in the path. The name 2-opt stems from reversing the middle subset, in effect removing 2 edges and reconnecting the subsets. Usually, all possible combinations are tested for improvements before the best path is returned, as illustrated in Figure 4.2.

```
Input: existingRoute
while(not stopping condition):
 DEFINE START
  best_distance = calculateDistance(existingRoute);
  for(each eligible node m):
   for(each eligible node n after m in path):
    newRoute = existingRoute[0...m-1]
    newRoute += reverse(existingRoute[m...n])
    newRoute += existingRoute[n+1...end]
    new_distance = calculateDistance(newRoute)
    if(new_distance < best_distance):
     existingRoute = newRoute
     GOTO START
return existingRoute
```

Figure 4.2: Pseudocode for 2-opt local search

Variable Neighbourhood Search

As described in Section 2.3.2.4: Metaheuristics for the Orienteering Problem, this search method applies several independent procedures sequentially, each examining a larger neighborhood than the previous, to improve a solution. In order to limit computational cost each procedure is applied only if the preceding one improved the solution, under the assumption that no improvements can be found far away if none could be found close to the initial solution.

A short description of each procedure, in the order performed, is given below. If an improvement is found the procedure is repeated until no more improvements are found before continuing with the next procedure. In this context a node is 'viable' to remove, interchange, or add if the operation does not invalidate the path according to the given constraints.

1. Interchange method: For each viable solution node interchange it with a viable unused node
2. Eliminate method: For each viable solution node eliminate it from the tour
3. Forward Insert method: For each viable solution node move it from its current posi-

tion in the path to a later viable position

4. Backward Insert method: Equivalent to Forward Insert method, for each viable solution node move it from its current position in the path to an earlier viable position
5. Swap method: For each pair of viable solution nodes interchange their positions
6. Add method: For each position in the path, insert a viable unused node

Given the cost $d(i,j,t)$ and reward $r(j)$ received for traversing the edge $v(i,j)$ at time t, the fitness function is expressed as:

$$f(s) = \frac{\sum_{c(i,j,t)\in s} r(j)}{\sum_{c(i,j,t)\in s} d(i,j,t)} \tag{4.9}$$

This fitness function assumes that decreasing the cost, usually at the expense of the reward, might lead to discovery of alternate paths resulting in higher end reward. Without this assumption the Eliminate method would never find improvements.

4.6 Solution Reinforcement

A fitness function calculates a single value, the relative goodness, of a solution or solution component given some goal. In ACO and IWDA fitness functions are used to evaluate the quality of a given solution, which is used to strengthen the trail parameters for the heuristic functions. Below, the specific procedures implemented in each algorithm are described in detail.

Intelligent Water Drops Algorithm

The IWDA has both a global and local soil update rule as described in Section 2.3.3.3: UpdateWaterDrop and Section 2.3.3.3: UpdateEdges.

During the solution construction procedure, each IWD moves sequentially one step at each

iterations, thus making the IWDs influence each other during each move. After each move the soil residing in the edge traversed is updated given the equation:

$$soil(i,j) = 1 - \epsilon_p * soil(i,j) - \epsilon_p * \Delta soil(i,j) \tag{4.10}$$

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s * time^2(i,j,vel^{IWD})}^{123} \tag{4.11}$$

From Equation 2.8 it is seen that the velocity of the IWD and the heuristic undesirability degree decides how much the given move should be reinforced. The parameters a_s, b_s, and c_s decide in which degree the heuristic should apply and ϵ_p controls the degree of exploration and exploitation.

After the construction phase is completed, the solution with the highest received reward is reinforced with Equation 2.9. This reinforcement is called the global update, and directs the search towards the most promising areas of the search space. The parameter p_{iwd}, also referred to as e_s, controls how much the IWD should affect the reinforcement, thus controlling the degree of exploration and exploitation.

Ant Colony Optimization Algorithm

The local pheromone update rule defined by Ant Colony System (ACS), described in Section 2.3.2.4: Ant Colony System, as well as the global pheromone update rule from Ant System, described in Section 2.3.2.3: ApplyPheromoneUpdate, were implemented.

The solution construction procedure is implemented such that ants build their solutions in parallel, the ants sequentially picking their next edge, such that when the local update rule is employed each selection affects all successive selections. As described in Section 2.3.2.4: Ant Colony System the local pheromone update rule degrades all traversed edges, encouraging exploration of unvisited subsets of the search space.

[1] a_s is a positive number, usually set to 1.0
[2] b_s is a small positive number, usually set to 0.01
[3] c_s is a positive number, usually set to 1.0

The global pheromone update rule encourages exploitation of the search space neighboring the best solution. The values are updated according to Equation 2.4, where the subset of solutions G_p was chosen to include all solutions found in the iteration, as well as the best found solution so far if not already included. The quality functions is implemented as

$$F(s) = \frac{f(s)}{\sum_{i \in G_p} f(i)} \tag{4.12}$$

where $f(s)$ is the sum of rewards of solution s, such that the quality values are normalized to the range from 0 to 1 and sums to 1. As the quality of a solution affects all the edges equally, an edge with high cost or leading to a low reward can still have a high pheromone value.

4.7 Solution Representation

A solution can be represented as an array of values mapping to nodes, where the order signifies the visitation order of the path. In this implementation, an integer array mapping to node indices in the state is used. As an example, a solution S given the set $I \in \{0,1,2,3,4\}$ of node indices could be represented as:

$$S = [0,2,4,1,0] \tag{4.13}$$

4.8 Applying the Model to the Simulator

The following sections will describe in detail how the environment of the International Aerial Robotics Competition mission 7a is modeled as a Time-Dependent Orienteering Problem with Time Windows, as well as functions essential to apply the model to the mission environment. Finally, environment characteristics not handled by the model is discussed.

4.8.1 Cost Function

Opposed to TDOPTW, the robots reverse their trajectory every 20 seconds in the competition, which needs to be considered in order to calculate the correct cost of traversing an edge. If the intersection time found by Equation 4.2 is lower than or equal to the remaining time until trajectory reversal, we can simply use this value to calculate edge cost. If not, the equation Equation 4.1 must be reformulated: We know the remaining time before the robot turns, and thus its position and resulting speed vector after turning.

Given

- U: The aerial robot's absolute speed
- V: The robot's speed vector after turning
- R: The aerial robot's position vector
- S: The robot's position vector at the point of turning
- T: The remaining time until trajectory reversal

we want to find the time where the positions of the robot and the aerial robot are equal

$$\begin{aligned} S + Vt &= R + U(t+T) \\ R - S + Vt &= U(t+T) \\ P + Vt &= U(t+T) \end{aligned} \tag{4.14}$$

With this, solving for t can be derived similarly to Equation 4.2.

4.8.2 Robot State

When reading the problem state, rewards, time windows, service times, and appropriate actions are not given explicitly and must be determined by the application. In order to determine these values, the current state of each robot is analyzed. The following states have been defined:

1. LEAVING: The robot will leave the arena erroneously before next trajectory reversal
2. PASSING_GREEN: The robot will leave the arena across the green line before the next trajectory reversal
3. TARGET_DIRECTION: The robot is heading towards the green line
4. OPPOSITE_DIRECTION: The robot is heading opposite of the green line
5. WRONG_DIRECTION: None of the conditions above apply

When more than one case applies to a robot, the state with the lowest index in the list above will have priority.

4.8.3 Rewards

Given the robot state as input, the following rewards are defined for each robot:

Robot state	Base reward	Positional reward	Maximum possible reward
LEAVING	100	No	100
PASSING_GREEN	0	No	0
TARGET_DIRECTION	5	Yes	63
OPPOSITE_DIRECTION	5	Yes	63
WRONG_DIRECTION	2.5	Yes	60.5

Table 4.1: Rewards attributed each robot, given the state and position of the robot. Positional reward ranges from 0 to 58, and the maximum possible reward denotes the highest final reward after the base reward and the positional reward have been summed

A robot that will leave the arena erroneously without interaction is awarded the maximum available reward, as repeated occurrences would result in failing the mission (and would penalize the team's score for each robot leaving). Robots that are assumed to pass the green line will receive no reward such that it will be ignored by the aerial robot. The robots heading towards or opposite from the green line is the easiest to guide over the green line, and will be slightly prioritized in order to simplify the problem state as quickly as possible. The positional reward, as illustrated in Figure 4.4, is added to the base reward for some states as defined in Table 4.1. Given the robot position (x, y), the arena

$x = [0,20], y = [0,20]$, and the green line $x = [0,20], y = 20$, the positional reward is defined by:

```
1 reward = 0
2 if(x < 4 or x > 16 or y < 4): reward += 25
3 if(y > 16): reward += 33
4 return reward
```

Figure 4.3: Pseudocode determining the positional rewards

The robots closest to the green line receives the highest positional reward in order to quickly get robots over the green line and simplify the problem state. Additionally, robots close to the sides are assumed to have a higher probability of leaving the arena erroneously, and will be given a higher reward than robots in the center of the arena. Rewards in overlapping areas (close to the green line and the sides) are cumulative. Robots positioned in the center of the arena are assumed to have a higher probability of colliding with obstacles and other robots (because all robots and obstacles are centered around the middle of the arena when the mission begins) and behaving unexpectedly, and thus receive no additional reward.

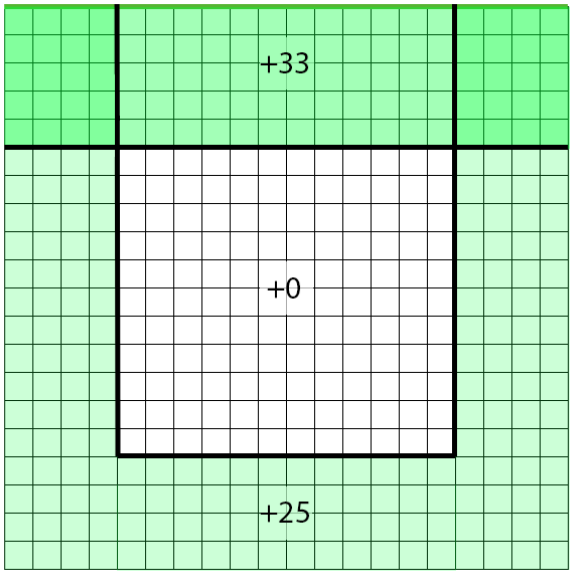

Figure 4.4: Shows the mapping of positional reward

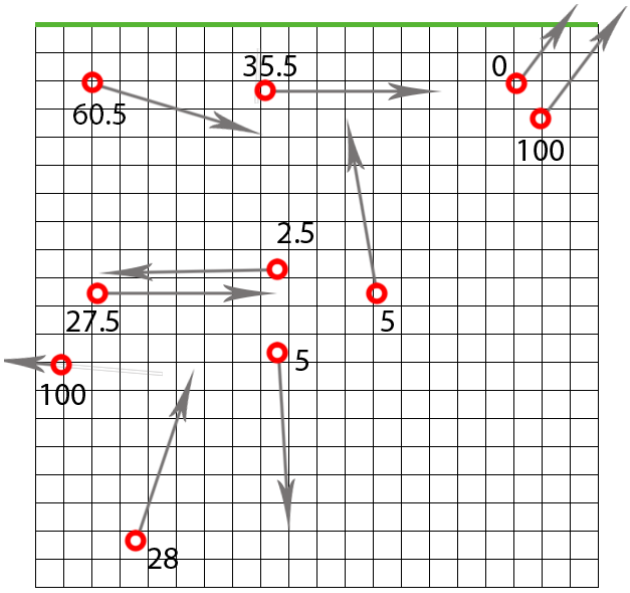

Figure 4.5: Shows an example of robots with their reward calculated from their position and direction

4.8.4 Time Windows

As denoted by *t(j)* and explained in Section 2.2.2: Time-Dependent Orienteering Problem with Time Windows, the time window of a robot represents the minimum and maximum cost that can be used to reach the robot to receive the reward. As the rewards defined in Section 4.8.3: Rewards are associated with a state, and also intuitively some goal state that should replace the current state, such as changing a 'LEAVING' state to a 'TARGET_DIRECTION' state, there are usually a limited time when it is useful to change this state. As an example, when the robot is leaving the arena it is not useful, or even possible, to turn it after it has left the arena, and the reward should only be able to be collected before that time. Below Table 4.2 defines the time windows associated with each robot state, given a cost budget of 40.

Robot state	Opening time	Closing time
LEAVING	0	(time until leaving arena)
PASSING_GREEN	0	0
TARGET_DIRECTION	(time remaining) - 6	(time remaining) + 17
OPPOSITE_DIRECTION	0, if (time remaining) >34, else	(time remaining) - 3, if (time remaining) >40, else
WRONG_DIRECTION	0	40

Table 4.2: The opening and closing time of a robot's time window, given the robot state and a cost budget of 40. 'Time remaining' denotes the time until trajectory reversal, and 'time until leaving arena' is the approximated value until the robot exits the arena. The state 'OPPOSITE_DIRECTION' has two possible time windows depending on the time until trajectory reversal

When a robot is leaving the arena, we can change its trajectory immediately, but intuitively no later than the time at which it leaves the arena. When the robot is expected to pass the green line it is beneficial to not interact with it, and the time window is never open.

When the robot is heading towards the green line without passing it, it is beneficial to turn it the opposite direction close to the trajectory reversal, such that it can benefit from 20 seconds of traveling toward the green line after the trajectory reversal. Correspondingly when the robot is heading opposite of the green line it is not beneficial to turn it close to the upwards trajectory reversal, and we have two cases: Early turn the robot towards the green line to benefit from traveling the correct direction most of the period, and late before the next downwards trajectory reversal turn it downwards to benefit from the following upwards trajectory reversal (equivalent to the time window of 'TARGET_DIRECTION'). It was found empirically that there were benefits of having a large time window for robots going the opposite direction, as this could lead to finding more beneficial solutions.

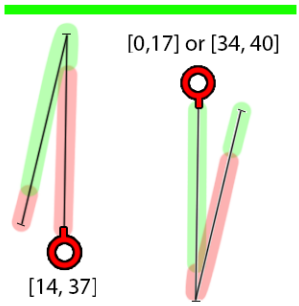

Figure 4.6: The green markings on the path illustrate when the node is available (i.e open) for interaction, while the red shows when the node is closed.

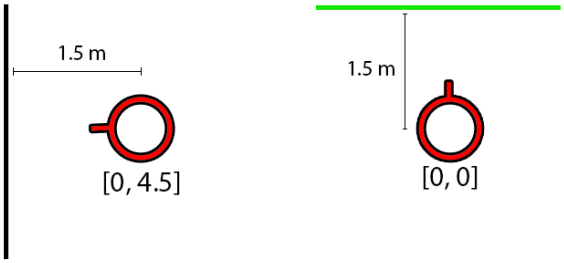

Figure 4.7: To the left the robot is assigned a closing window equal to the time it uses to leave the arena erroneously, and to the right it is set to zero because it leaves the arena correctly

4.8.5 Service Time

The service time is approximated based on the initial state, as shown in Table 4.3, based on the available actions. Each individual action takes 4 seconds to perform, such that turning a robot 45 degrees three times takes 12 seconds.

Robot state	Assumed action plan	Service time
LEAVING	createTurnPlan(...)	{4, 8, 12}
PASSING_GREEN	(Empty)	0
TARGET_DIRECTION	180 degrees	4
OPPOSITE_DIRECTION	180 degrees	4
WRONG_DIRECTION	createTurnPlan(...)	{4, 8, 12}

Table 4.3: Service costs and assumed action plan given the robot state. The function 'createTurnPlan' calculates the number of actions required to turn the robot towards the green line, outputting one of three values

Because the trajectory noise cannot be predicted, the environment is assumed to be deterministic in these approximations. Thus, given the action selection defined in Section 4.8.6: Action Selection, robots heading towards or opposite from the green line will only require a single turn of 180 degrees. For robots having other angles, the number of actions required needs to be calculated, using the function denoted in Table 4.3 as 'createTurnPlan',

which returns minimally 1 and maximally 3 actions to be performed for a robot heading the wrong direction, resulting in a service time in $\{4, 8, 12\}$.

4.8.6 Action Selection

As described in Section 2.1.2.1: Targets there are two possible robot interactions:

- Land in front (causing collision): Turns robot 180 degrees
- Land on top: Turns robot 45 degrees clockwise

The aerial robot will create an action plan for the robot upon arrival that minimizes the number of actions required to turn the robot as directly as possible towards the green line, or opposite if less than 7 seconds remain until trajectory reversal such that it will shortly turn towards the green line. Given the minimum increment of 45 degrees, the action plan must be created so that the robot crosses the green line before possibly crossing the sides.

4.8.7 Commitment to Plan

As the state of the mission arena changes continuously and heuristic search algorithms are not guaranteed to find the optimal solution, ACO and IWDA are expected to often produce slightly different solutions with no or relatively small changes in the state. On one hand the algorithms might in a short time span find solutions with insignificant increase in reward, such that strictly following the plan with highest reward might lead to changing plans frequently without any significant benefit in the mission. On the other hand, small fluctuations in the state might yield fluctuations in a solution's reward as well, such that two similar solutions might alternate as the perceived best solution.

In order to complete the mission it is crucial that the aerial robot is able to complete actions, preferably most of the plan as a plan with high total reward may still have low rewards early in the plan, and should not change plans unless it has a large benefit. Such a case could be, given the stochastic environment, that a previously safe robot is now sud-

denly about to leave the arena and requires saving. Thus some strategy is required to decide when to commit to and when to change the currently active plan.

In order for a competing solution to replace the current solution it is required to be exponentially better, given a small constant as the exponent. Given that $r_{current}$ is the reward of the current active solution $s_{current}$ and r_{new} the reward of the competing solution s_{new}, the strategy can be expressed as:

$$s_{current} = \begin{cases} s_{new} & if\, r_{new} > r_{current}^{e} \\ s_{current} & else \end{cases} \tag{4.15}$$

for some constant e. In the implemented strategy, a constant of $e = 1.3$ was used. It is a simple technique, but highly dependent on the allocation of rewards.

4.8.8 Solving Additional Requirements

As mentioned in Section 2.2.2.1: Problems Not Covered by TDOPTW the environment of TDOPTW is fully observable, discrete, and deterministic. In the competition and simulator the environment is stochastic and continuous, which needs to be handled and is discussed below. The partial observability will not be solved in this dissertation due to uncertainties regarding the capabilities of the aerial robot's sensory equipment.

Stochasticity

The stochasticity of the mission is made impossible to plan for or predict for longer periods of time, as the trajectory noise occurring every 5 seconds turns the robots 0 to 20 degrees in the clockwise or counterclockwise direction. Fortunately, experience shows that during traversal of an edge this has minor significance on the general state of the arena, such that the aerial robot should often be able to complete its current task. Thus, the best option is to use the median values as if they were deterministic.

In order to mitigate the stochasticity for longer periods of time, the only solution is to review the state and plan often in order to account for recent changes, which IWDA and ACO is suited for because of their fast solution generation. In the implementation the state is reviewed and a new plan is created every 500 milliseconds. This brings up another problem, namely commitment to the current plan.

Although the aerial robot needs to be prepared to switch plan if rewards change drastically, e.g. in the event of a previously safe robot leaving the arena, the worst that can happen is that the aerial robot switches plan every fraction of a second because of small changes in the state, travel cost, or reward, or in the case of not being able to find the optimal solution multiple times in a row. The solution is, when comparing the reward to be collected in each plan, to add a margin to the current plan, such that the new plan needs to be significantly better than the current in order to become the active plan, as discussed in Section 4.8.7: Commitment to Plan.

Continuity

Because of the stochastic environment the positions and trajectories of the robots cannot be predicted accurately, and there are infinitely many possibilities. Because of small deviations between the state the solution was created from and the current state, the application is prone to perform wasteful actions. This is reduced by frequent replanning as discussed before, and handing control of the aerial robot's movement over to a separate controller using real-time information when close to the targeted robot.

4.9 Greedy Controller

A greedy algorithm is an algorithm that selects the best choice available at the moment, without regard of possible future consequences Cormen et al. (2009). The greedy controller was implemented to evaluate the performance of applying TDOPTW to the mission, and regards the ground robot closest to the green line as the most important object to handle.

When the closest robot is found, the aerial robot will follow this robot and guide the robot until it crosses the green line. Then it will find the next robot closest to the green line and continue until there are no robots left.

The greedy controller is the simplest form of decision making for the aerial robot to implement. It does not require much to work, except from finding a ground robot, and the ability to land on and in front of a ground robot. It is assumed that Ascend NTNU will use this implementation for the first physical test on the actual mission due to its simplicity.

4.10 Summary

Opposed to the only known study of TDOPTW, which represented time-dependent cost as a matrix of costs given all possible interaction timings, the implementation proposed in this dissertation use the Euclidean distance between points moving as a function of time. Each cost is calculated dynamically when required, and although it would be possible to expand the initial graph with each possible edge cost between each pair of nodes, this would lead to graphs magnitudes larger than the initial graph and be computationally expensive to search through. One possible consequence of choosing this model is that the trail parameters, since they can depend on multiple costs, can misrepresent the benefits an edge provides to a specific solution.

In order to select the next edge to add to a solution, both ACO and IWDA calculate the heuristic value of the edge based on the reward and cost it provides, such that shorter edges leading to higher rewards are preferred. The heuristic value and trail parameter of an edge are used in a selection method to calculate the transition probability to that edge, in other words the odds of adding the edge to the solution. Rank selection and tournament selection was applied to ACO alongside the default fitness proportionate selection. IWDA only implemented fitness proportionate selection, because rank selection has been shown to yield only minor improvements with a high computational cost with IWDA. In order to avoid searching through infeasible solutions, the search space is dynamically pruned of invalid solution components. Both 2-opt and Variable Neighborhood Search has been im-

plemented with ACO as local search algorithms, to see if a tour improvement procedure could improve the performance of the tour construction procedure.

Both IWDA and ACO employ a local and global solution reinforcement procedure. In IWDA the local reinforcement immediately removes some soil after an IWD has traversed an edge, and the global reinforcement is performed after each iteration by removing some soil from the edges of the iteration best solution, based on the amount of soil in the IWD. In ACO the local reinforcement immediately removes some pheromone from the edge after an ant has traversed it, and the global reinforcement is performed after each iteration by adding pheromone to the so far best solution. The solutions are represented as an array of integers, where each integer represents a node.

The state retrieved from the simulator is converted to a TDOPTW model by assessing the current and future position of each robot. Using the positions and remaining time until trajectory reversal, rewards and time windows are assigned to nodes according to a static map. Service time is calculated based on the initial trajectory of each node in order to turn it towards the green line. The action plan, which defines the cheapest way to turn the robot towards the green line, is created upon visiting the robot. While plans are calculated every 500ms by the controller the commitment strategy decides if the currently active plan should be exchanged with the new plan, based on the rewards of the solutions.

TDOPTW does not regard the partially observable, continuous, and stochastic properties of the environment. While the simulator provides a fully observable environment, unlike the mission, stochasticity and continuity must be handled. By ignoring noise when modeling the state and frequently replanning for the aerial robot, the problems associated with stochasticity and continuity will hopefully be solved, or at least mitigated.

In order to evaluate the performance of the TDOPTW controller for the simulator, a greedy controller always guiding the robot closest to the green line, possibly the simplest approach to solving the mission, was implemented.

Chapter 5

Results and discussion

In this chapter the results from testing the implementation will be shown and discussed. As the construction of the physical aerial robot was not completed at the time this dissertation was written, only benchmark tests and simulator tests have been performed.

Unless stated otherwise, the parameters used for IWDA are listed in Table 5.1 and for ACO in Table 5.2. The parameter values were based on the literature investigated in Section 2.3.3: Intelligent Water Drops and Section 2.3.2: Ant Colony Optimization respectively, and adjusted to maximize preliminary benchmark results on the same test sets described in Section 5.1.2: Time-Dependent Orienteering Problem with Time Windows.

Parameter	Value
m (number of agents)	50
Number of iterations	100
Initial soil	1000
Initial velocity	4
a_v	1
b_v	0.01
c_v	1
a_s	1
b_s	0.01
c_s	1
e_s	0.1
e_p	0.1

Table 5.1: Default parameters for IWDA

Parameter	Value
m (number of agents)	20
Number of iterations	10
t_0 (initial pheromone)	0.001
a (pheromone importance)	1
b (heuristic importance)	1
p (evaporation rate)	0.5
u_0 (exploitation probability)	0.9

Table 5.2: Default parameters for ACO

5.1 Benchmark Tests

Some test instances exists for TDOPTW, created by Garcia et al. (2010), but as described in Section 4.1: Representing Time-Dependent Cost they were not suitable for the implementation targeting the mission environment. Thus, in order to compare the performance of ACO and IWDA with the literature, additional tests were performed on OPTW. To the extent of the authors' knowledge, IWDA has not been tested on any variations of the Orienteering Problem, and therefore it was of importance to investigate its performance.

Comparing ACO using rank, tournament, and fitness proportionate selection no significant

differences were found (the results can be found in Table A.3, Table A.4, and Table 5.11 respectively). Since fitness proportionate selection is typically used with ACO, and IWDA use it as well, it was decided to continue testing using only this selection method.

Two local search methods have been implemented with ACO, namely 2-opt and Variable Neighborhood Search, both described in Section 4.5: Local Search. Preliminary results showed insignificant differences in received reward and significantly greater computational cost using Variable Neighborhood Search, and thus benchmark tests will only be carried out using 2-opt as the local search procedure.

In Table 5.3 the different symbols and abbreviations used in the following results are described.

Symbol/abbreviation	Description
Name	The problem instance name excluding the prefix '50_'
Max Reward	Maximum solution reward
Avg Reward	Average solution reward
Min Reward	Minimum solution reward
Dev Reward	Standard deviation of solution rewards
Max Cost	Maximum solution cost
Avg Cost	Average solution cost
Min Cost	Minimum solution cost
Dev Cost	Standard deviation of solution costs
Max Time	Maximum problem instance computation time
Avg Time	Average problem instance computation time
Min Time	Minimum problem instance computation time
Dev Time	Standard deviation of problem instance computation times
Max Itr	Maximum number of algorithm iterations to find best solution
Avg Itr	Average number of algorithm iterations to find best solution
Min Itr	Minimum number of algorithm iterations to find best solution
Dev Itr	Standard deviation of number of iterations to find best solution

Table 5.3: Description of abbreviations and symbols used in describing the benchmark results

The following two subsections describes the results of a 100 runs on six different problem instances for ACO and IWDA.

5.1.1 Orienteering Problem with Time Windows

The problem instances used was found on http://www.mech.kuleuven.be, created in relation to Righini and Salani (2006). From the collection specific test instances were chosen to have high variation in cost budgets, time windows, and node positions, especially considering the degree of clustering of nodes. Each of the problem instances has 50 nodes to visit, with a given start node and end node.

Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Cost Budget	Max Cost	Avg Cost	Min Cost	Dev cost
c101	210.00	166.40	140.00	13.00	1236	1,232.54	1,181.06	1,148.33	25.14
c109	270.00	225.50	200.00	13.44	1236	1,234.17	1,169.82	1,102.33	35.54
r107	203.00	176.77	153.00	9.97	230	229.80	221.13	195.72	7.55
r109	147.00	123.69	109.00	8.33	230	224.20	201.67	183.49	10.57
rc101	170.00	148.00	130.00	8.49	240	226.04	209.44	189.87	6.21
rc106	190.00	160.50	140.00	12.28	240	233.95	208.79	191.15	11.86

Table 5.4: Reward and cost results for IWD with low exploitation

Name	Max Time	Avg Time	Min Time	Dev time	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	1.3104	1.1914	1.0788	0.0516	99.00	33.00	1.00	28.72
c109	2.1613	2.0476	1.9794	0.0317	100.00	33.00	1.00	24.39
r107	2.1866	2.0206	1.9559	0.0386	94.00	57.00	9.00	18.51
r109	1.3567	1.2278	1.1231	0.0570	100.00	41.00	1.00	28.78
rc101	1.0580	0.9361	0.8493	0.0475	99.00	43.00	1.00	28.77
rc106	1.1823	1.1307	1.0773	0.0202	98.00	46.00	1.00	29.75

Table 5.5: Time and iteration results for IWD with low exploitation

Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Cost Budget	Max Cost	Avg Cost	Min Cost	Dev cost
c101	260.00	260.00	260.00	0.00	1236	1,174.50	1,169.41	1,169.20	1.04
c109	330.00	322.00	320.00	4.00	1236	1,233.80	1,210.89	1,181.00	15.95
r107	204.00	199.39	196.00	2.16	230	226.20	225.83	224.40	0.62
r109	186.00	183.60	180.00	2.94	230	221.90	216.18	207.60	7.01
rc101	170.00	170.00	170.00	0.00	240	219.20	218.66	216.20	1.15
rc106	200.00	200.00	200.00	0.00	240	234.40	225.13	219.20	5.12

Table 5.6: Reward and cost results for ACO using fitness proportionate selection and no local search

Name	Max Time	Avg Time	Min Time	Dev time	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	0.0813	0.0670	0.0622	0.0038	8.00	3.00	1.00	2.51
c109	0.1048	0.0964	0.0895	0.0040	8.00	3.00	1.00	2.28
r107	0.0824	0.0720	0.0668	0.0035	8.00	6.00	4.00	1.23
r109	0.0721	0.0607	0.0556	0.0035	10.00	5.00	2.00	3.44
rc101	0.0510	0.0418	0.0382	0.0032	3.00	1.00	1.00	0.97
rc106	0.0620	0.0528	0.0488	0.0035	4.00	2.00	1.00	1.08

Table 5.7: Time and iteration results for ACO using fitness proportionate selection and no local search

Although no previous applications of IWDA on the problem has been found, the results show that both ACO and IWDA can find solutions significantly better than the worst solutions encountered. The computation time used to find these solutions are low compared to what could be expected from a brute force search of a graph this size.

Name	Best known	ACO	IWDA
c101	320.00	260.00	210.00
c109	380.00	330.00	270.00
r107	299.00	204.00	203.00
r109	277.00	186.00	147.00
rc101	219.00	170.00	170.00
rc106	252.00	200.00	190.00

Table 5.8: Comparison with best known results

Table 5.8 compares the achieved maximum reward with the best known results, as reported by Gunawan et al. (2015). Neither ACO nor IWDA are able to find the best known solutions for any of the problem instances, but finding optimal solutions to such problems requires lots of fine-tuning and optimization of the algorithms towards the specific problem instance and the general problem. The reason for this is the narrow space that the optimal solutions exist in.

In every graph search there are neighborhoods of solutions that are similar in composition, but with the given constraints the neighborhood of excellent solutions might be very bad, or in worst case, consisting of mostly illegal solutions. The cost budget and time windows

might eliminate solutions in the neighborhood of optimal solutions, while the reward and cost of getting there might misguide heuristic algorithms away from these neighborhoods.

5.1.2 Time-Dependent Orienteering Problem with Time Windows

The problem instances used were based on the problem instances used in Section 5.1.1: Orienteering Problem with Time Windows, adopted by inserting node values not considered in OPTW: For each node a random trajectory value in the range $[0, 2\pi)$, and a speed of **0.33** (as the robot speed in the mission is 0.33 m/s). There are no known optima for these test instances, and so the performance can only be evaluated between ACO and IWDA.

Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Cost Budget	Max Cost	Avg Cost	Min Cost	Dev cost
c101	160.00	132.90	110.00	8.87	1236	897.68	739.39	653.73	49.26
c109	180.00	150.70	110.00	15.31	1236	1,076.08	873.63	712.13	76.87
r107	145.00	120.21	97.00	9.62	230	219.92	180.64	153.70	14.17
r109	143.00	119.24	102.00	9.92	230	185.37	151.27	128.53	11.45
rc101	160.00	136.10	100.00	11.04	240	212.53	189.74	145.18	12.04
rc106	150.00	123.80	100.00	10.66	240	195.50	173.05	141.59	14.87

Table 5.9: Reward and cost results for IWD with low exploitation

Name	Max Time	Avg Time	Min Time	Dev time	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	2.1841	2.0591	1.9849	0.0460	99.00	42.00	1.00	27.81
c109	2.6679	2.5725	2.4822	0.0356	100.00	44.00	2.00	30.59
r107	2.2776	2.1375	2.0576	0.0410	100.00	35.00	1.00	28.01
r109	1.9883	1.9181	1.8297	0.0361	99.00	20.00	1.00	21.36
rc101	1.4162	1.3394	1.2456	0.0276	97.00	25.00	1.00	26.84
rc106	1.5268	1.4602	1.4046	0.0278	93.00	28.00	1.00	22.03

Table 5.10: Time and iteration results for IWD with low exploitation

Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Cost Budget	Max Cost	Avg Cost	Min Cost	Dev cost
c101	130.00	130.00	130.00	0.00	1236	858.93	858.93	858.93	0.00
c109	210.00	204.00	200.00	4.90	1236	1,229.39	1,186.45	1,088.19	45.39
r107	155.00	149.13	148.00	1.79	230	227.64	216.02	208.22	6.17
r109	154.00	146.97	138.00	6.14	230	220.18	209.58	192.38	6.94
rc101	150.00	148.70	140.00	3.36	240	237.41	231.73	228.32	4.27
rc106	150.00	150.00	150.00	0.00	240	226.10	223.14	221.61	2.08

Table 5.11: Reward and cost results for ACO without local search

Name	Max Time	Avg Time	Min Time	Dev time	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	0.0831	0.0721	0.0654	0.0039	1.00	1.00	1.00	0.00
c109	0.1304	0.1168	0.1064	0.0056	10.00	4.00	1.00	3.04
r107	0.1285	0.1150	0.1060	0.0052	9.00	3.00	1.00	2.81
r109	0.1283	0.1066	0.0979	0.0054	10.00	4.00	1.00	3.27
rc101	0.0876	0.0759	0.0693	0.0038	3.00	1.00	1.00	0.50
rc106	0.0933	0.0842	0.0753	0.0042	8.00	5.00	2.00	1.95

Table 5.12: Time and iteration results for ACO without local search

The results show that both ACO and IWDA, neither of which have been shown to solve the problem before, find solutions significantly better than the worst solutions encountered. The differences between the two algorithms are smaller than on OPTW, and IWDA had the best results on two problem instances.

The difficulty discussed earlier with finding good solutions in bad neighborhoods is emphasized by the time-dependent cost of the problems, as the neighborhoods change depending on the cost used to get there. Furthermore similar solutions might differ more in both cost and reward than in OPTW, and the trails of the algorithms might not be fully utilized, as discussed in Section 4.1: Representing Time-Dependent Cost.

For the purpose of solving the mission it is assumed that it is more important to generally find a high average reward and low deviation in rewards rather than a high maximum reward, as the state of the problem changes significantly throughout the mission. This will be further discussed in Section 5.2: Simulator.

	Without local search			With local search		
Name	Max Reward	Avg Reward	Avg Time	Max Reward	Avg Reward	Avg Time
c101	130.00	130.00	0.0721	130.00	130.00	0.0797
c109	210.00	204.00	0.1168	210.00	203.40	0.1352
r107	155.00	149.13	0.1150	150.00	149.07	0.1412
r109	154.00	146.97	0.1066	153.00	148.03	0.1344
rc101	150.00	148.70	0.0759	150.00	150.00	0.0932
rc106	150.00	150.00	0.0842	150.00	148.90	0.0980

Table 5.13: Comparing ACO with 2-opt local search and without local search

As mentioned in Section 2.3.2.3: LocalSearch a local search procedure (i.e. tour improvement procedure) can highly benefit tour construction procedures when creating solutions, but as mentioned in Section 2.2.3.3: Orienteering Problem with Time Windows, creating an efficient local search procedure for problems with time windows is difficult. 2-opt, as described in Section 4.5: Local Search has been tested with ACO with the results shown in Table 5.13. The difference in received rewards are insignificant given statistical error, and it was observed that the local search procedure generally were unable to improve any solutions because of the tight constraints. In light of the increased computational cost of the local search, remaining tests will be performed without the use of a local search procedure.

	High exploitation			Low exploitation			
Name	Max Reward	Avg Reward	Avg Itr	Max Reward	Avg Reward	Dev Reward	Avg Itr
c101	170.00	112.60	10.00	160.00	132.90	8.87	42.00
c109	180.00	138.30	24.00	180.00	150.70	15.31	44.00
r107	142.00	111.95	8.00	145.00	120.21	9.62	35.00
r109	142.00	111.32	11.00	143.00	119.24	9.92	20.00
rc101	150.00	115.40	2.00	160.00	136.10	11.04	25.00
rc106	140.00	106.20	3.00	150.00	123.80	10.66	28.00

Table 5.14: Comparing IWD with high and low exploitation

As described in Section 2.3.3: Intelligent Water Drops IWDA changes its search space by removing soil from rivers, but it is not able to replace the soil. So if the algorithm finds a good neighborhood it is difficult for it to find solutions in other neighborhoods, and it gets worse the longer the so far best solution has been reinforced.

As can be seen in Figure 5.1, a neighborhood which is available if an IWD moves to ‘B’, it

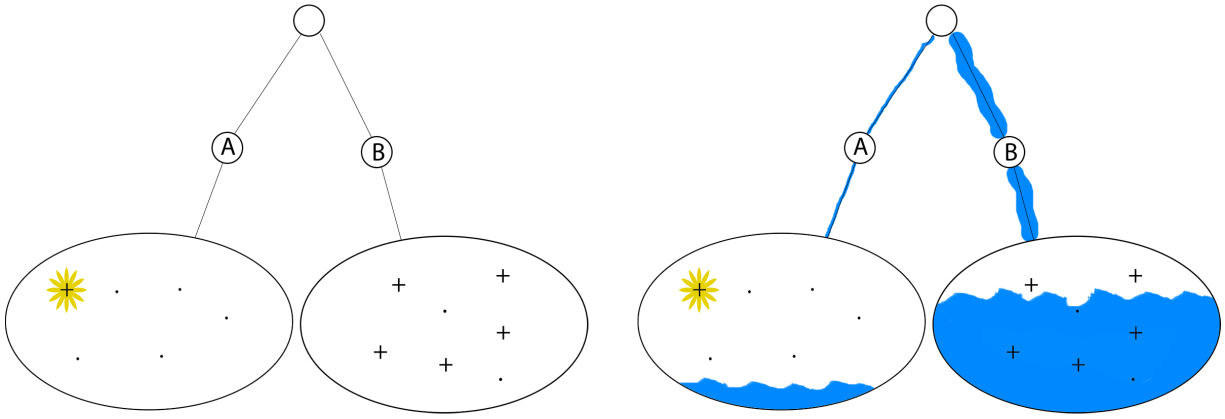

Figure 5.1: A local good neighborhood will attract more IWDs, thus higher search pressure, while the global solution may be in another neighborhood.

is able to find solutions which are locally good, but not optimal as designated by the yellow star. The global optimal solution is in the neighborhood that opens up if 'A' is visited, but the algorithm is prone to continue reinforcing the path to B and put the pressure on that neighborhood, which makes it hard to find the optimal solution. But there will always be a probability for an IWD to choose A, and that probability depends on the amount of soil residing in the edges. So in order to lower the exploitation pressure of the algorithm, the parameters for removing soil during traversal and solution reinforcement, respectively e_p and e_s, can be decreased in order to decrease the convergence speed.

Table 5.14 compares two different parameter settings for IWDA, one with high ($e_s = e_p = 0.9$) and one with low exploitation pressure ($e_s = e_p = 0.1$). The results show that lowering the degree of exploitation allows the algorithm to discover better solutions with the same number of iterations as the upper limit, i.e. with the same computational cost. Also, we see increased average number of iterations used to find the best solution with lower exploitation, which illustrates the lower convergence speed.

In the following diagrams, some interesting results from the benchmark tests on ACO and IWDA are compared side by side.

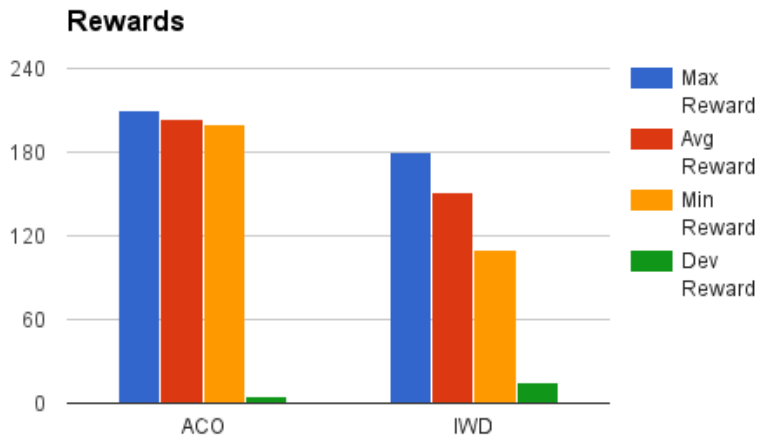

Figure 5.2: Reward received on problem instance 'c109'

Figure 5.2 shows that ACO finds higher reward on problem instance 'c109' for TDOPTW. The average and minimum reward found in ACO is higher than the maximum reward found in IWDA, and has low deviation, which illustrates how stable it is.

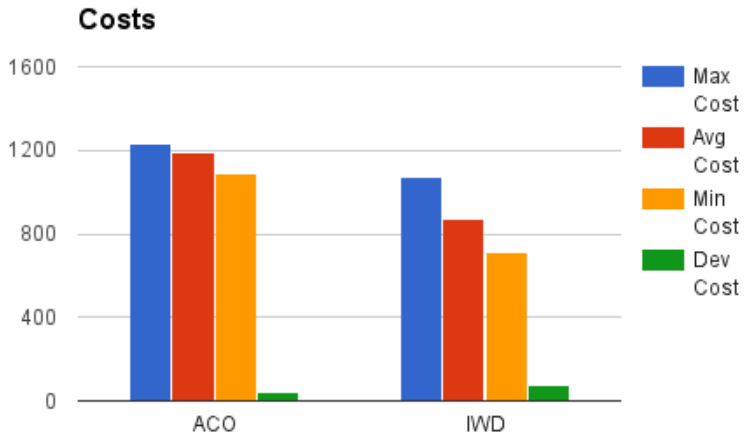

Figure 5.3: Cost spent on problem instance 'c109'

Figure 5.3 shows that ACO use more cost, and thus as seen in Figure 5.2 is able to find higher rewards as well. As the cost budget is 1236, it is apparent that IWDA stagnates in neighborhoods where it is too far to other nodes with open time windows. As ACO has a

smaller deviation in rewards than IWDA, it is not unexpected that it has a smaller deviation in costs as well.

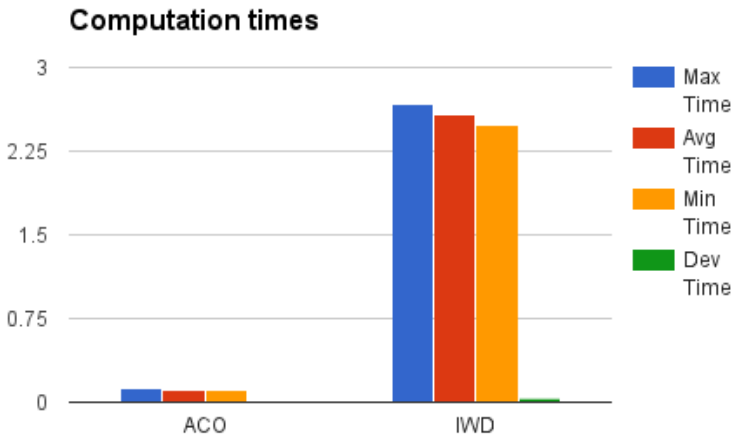

Figure 5.4: Computation time in seconds spent on problem instance 'c109'

In Figure 5.4 the difference in computational cost is illustrated. Given that IWDA have a higher number of agents and iterations to find solutions, it is to be expected. The deviation can be explained by difference in solution lengths explored, where creating longer solutions require additional computations to be performed. Since ACO has an almost insignificant deviation in its computational cost (too small to be shown in the figure, but present in Table 5.12), it can be assumed that there are relatively small variations in the explored search space between each run. These results show that using rank selection in IWDA, as mentioned in Section 2.3.3.4: Modification of Selection Method, would make the algorithm infeasible for use in the mission because of high computational cost.

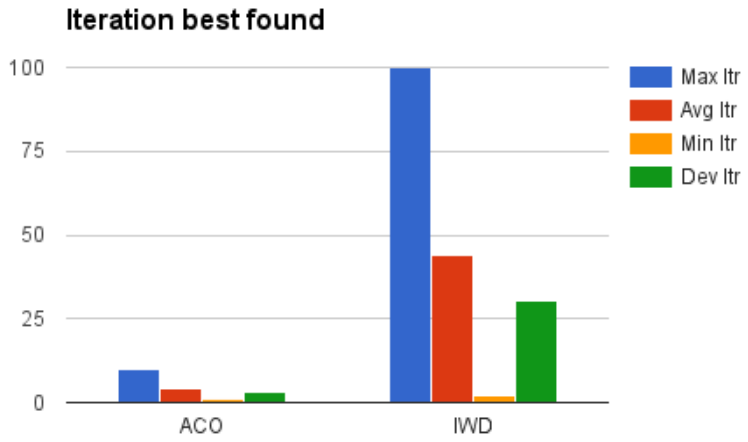

Figure 5.5: Iteration when best solution was found on problem instance 'c109'

Knowing that the algorithms run a different number of iterations, the algorithms need to be compared relatively in Figure 5.5. The difference between the maximum and minimum number of iterations, and the high deviation, illustrates how edge selection, based on the transition probabilities of the edges, can affect how quickly the best solutions are found, and that the algorithms can escape bad subspaces to find good solutions later. The averages are low compared to the maximum values, which shows that it is possible to trade lower computational cost for slightly worse solutions on average, and vice versa. This is supported by 'convergence in value' as discussed in Section 2.3.3.4: Convergence Properties of Intelligent Water Drops Algorithm, where IWDA is said to find the optimal solution given enough iterations.

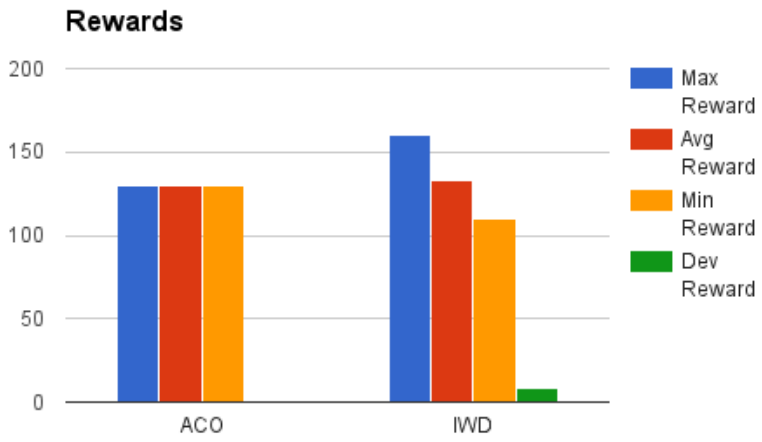

Figure 5.6: Reward received on problem instance ‘c101’

Problem instance ‘c101’ was one of two problem instances where IWDA achieved better rewards than ACO. In Figure 5.6 it is shown that the average reward of IWDA is higher than the maximum reward found by ACO on ‘c101’. As seen in Table 5.12 ACO usually requires few iterations to find its best solution, which indicates that it gets stuck in a local optimum and is unable to escape during the allocated iterations. An important aspect to notice is that in the instances where IWDA has higher maximum reward than ACO, the average of ACO is still close to or higher than that of IWDA, and with no deviation it could be preferable to use ACO in the mission because of its consistency. Producing equal or similar solutions for similar states reduces the need of a complex commitment algorithm.

5.2 Simulator

All of the tests in this section were run in the implemented simulator and recorded. Each test ran until there were no more ground robots left to guide, even if the simulation exceeded 10 minutes. The asterisk symbol (*) is used to illustrate that the robot left the arena erroneously at the given time.

As indicated by the results in the previous section, IWDA had a high deviation in solution quality. As a result of the high difference in solutions, the controller had problems with committing to a specific plan to follow. Thus the IWDA controller used a significant amount of time flying back and forth without handling robots, as illustrated in Figure 5.7: In step 1 a plan is active; 3 seconds later, in step 2, the plan has changed; before the aerial robot is able to move a significant distance (as seen in step 3), in step 4 a new plan similar to the one in step 1 is set as active, only 3 seconds after the last replanning. This behavior is prevalent with IWDA and explains why such a long time is required to guide all robots over the green line. Given the evidence and the significant modifications required to resolve the commitment issue with the IWDA controller it was decided of stop the simulator testing with IWDA after 3 runs and continue only with ACO.

Run	Min. requirement	All requirements	Lost robots	1. robot	2. robot	3. robot	4. robot	5. robot	6. robot	7. robot	8. robot	9. robot	10. robot
1	473	593	0	38	119	156	240	276	454	473	515	592	593
2	496	678	0	79	176	231	350	380	412	496	573	587	678
3	458	695	0	38	150	320	357	398	415	458	514	532	695
Avg	475,67	655,33	0,00	51,67	148,33	235,67	315,67	351,33	427,00	475,67	534,00	570,33	655,33

Table 5.15: TDOPTW with IWDA Controller

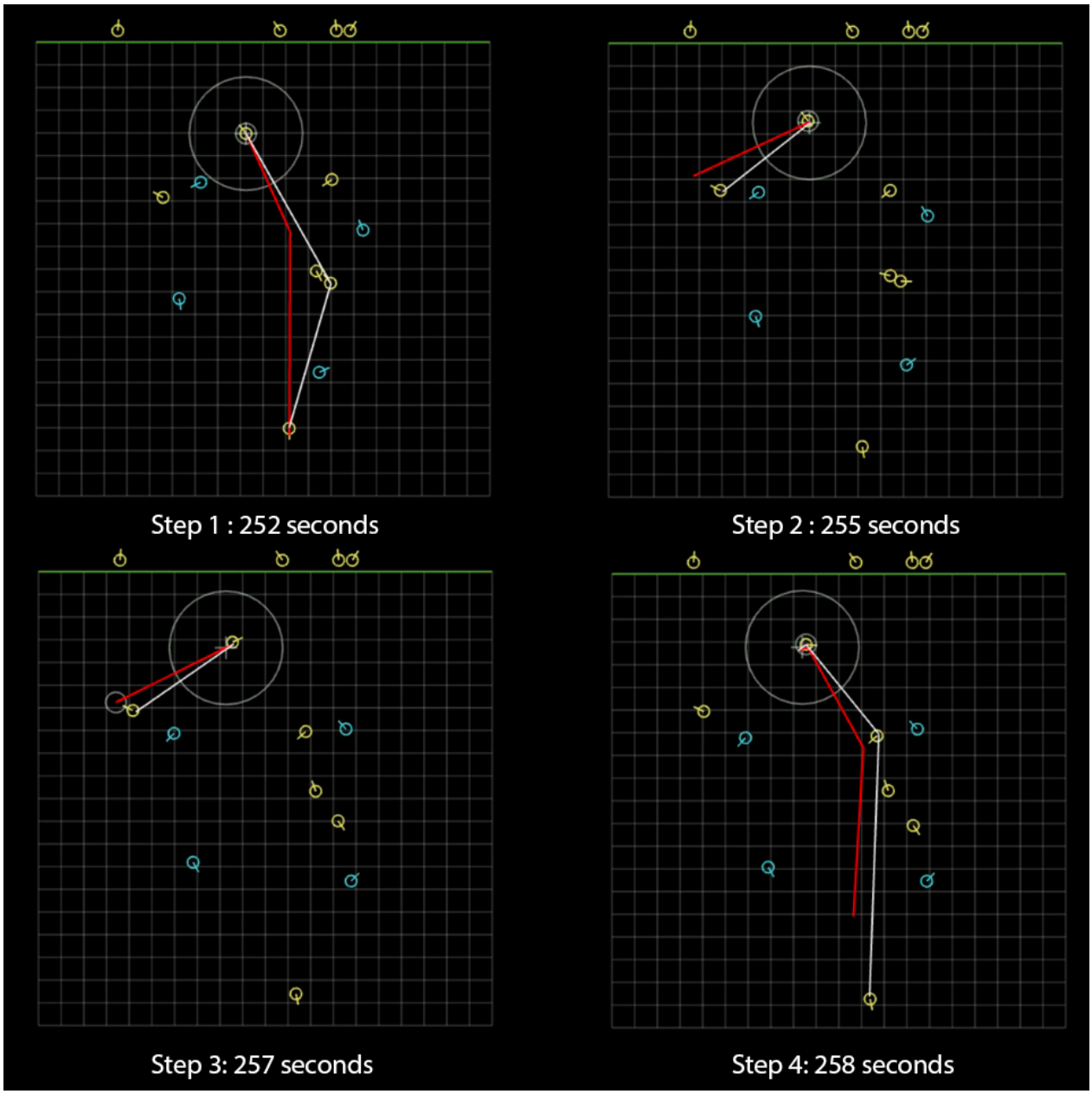

Figure 5.7: Shows a picture series of the TDOPTW controller using IWDA ranging over 8 seconds, which illustrates how the IWDA controller struggles with commitment to a path

5.2.1 Behavior of Greedy Controller

Run	Min. requirement	All requirements	Lost robots	1. robot	2. robot	3. robot	4. robot	5. robot	6. robot	7. robot	8. robot	9. robot	10. robot
1	418	-	1	37	73	120	179	240	313*	353	418	509	620
2	387	720	0	36	77	107	168	233	307	387	527	627	720
3	378	619	0	39	86	151	175	227	278	378	460	530	619
4	418	-	2	36	78	127	153	200	314	376*	418	419*	532
5	371	628	0	35	71	105	189	220	275	371	437	540	628
6	439	713	0	38	88	141	250	293	372	439	546	628	713
7	336	674	0	36	67	97	158	233	306	336	419	553	674
8	411	628	0	38	75	113	176	259	337	411	470	536	628
9	300	-	1	51	75	111	151*	158	173	237	300	373	478
10	348	559	0	50	78	117	159	212	247	348	415	488	559
Avg	380,60	648,71	0,40	39,60	76,80	118,90	178,56	227,50	289,89	362,22	441,00	531,56	617,10

Table 5.16: Greedy Controller, where the asterisk denotes that the robot left the arena erroneously

As seen from the results in table Table 5.16 in all 10 runs the greedy algorithm achieves the minimum requirement of the mission to guide 7 robots over green line under 10 minutes, but never to guide all 10 robots over the green line under 10 minutes. Figure 5.2.1 compares the table data from the greedy runs, and all runs are relatively close to the average. As the aerial robot only guides a single robot at the time it does not induce any additional uncertainties in the state, opposed to TDOPTW, which is discussed later.

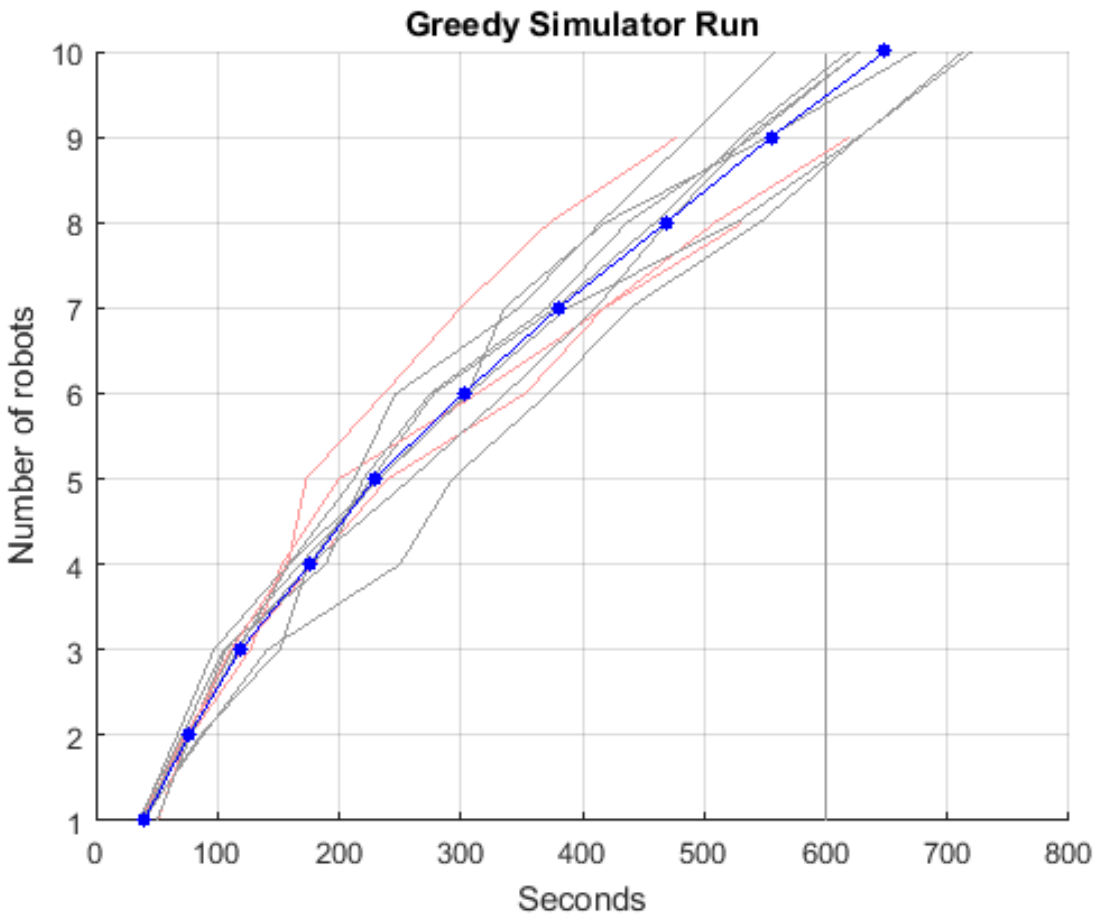

Figure 5.8: Shows the result of 10 different runs using the greedy controller. The blue line is the average, and the red lines show runs were robots were lost

As the greedy controller finds the robot closest to the green line and handles this robot until it has crossed the green line, it uses relatively short time, average of 40 seconds, to guide the first robot. As seen from Figure 5.2.1 it takes longer and longer to guide robots over the green line, and the average line flattens out. This can be explained by the fact that the robots starts moving outwards from a circle, and the nature of the greedy algorithm shines through and will not handle the robots furthest away until its the only one left to handle.

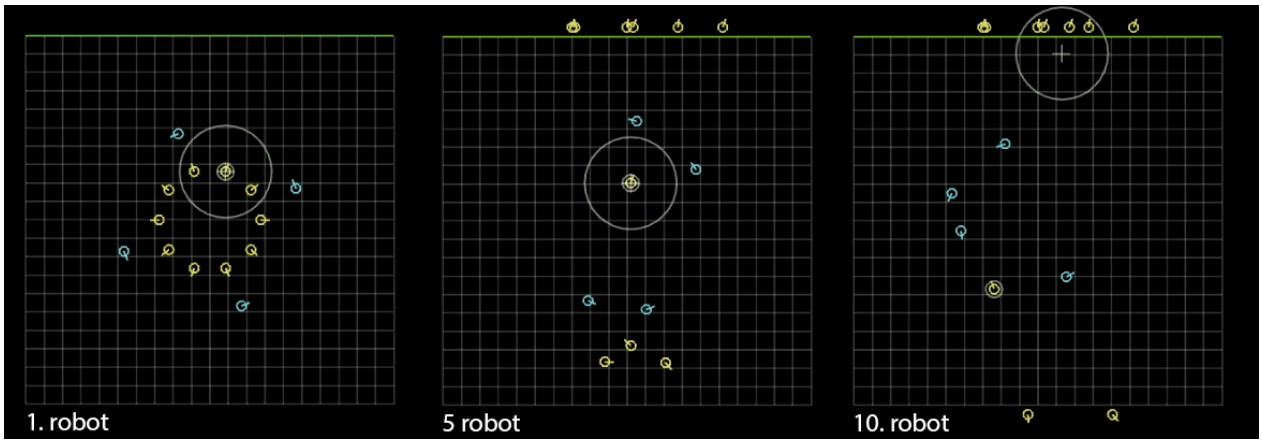

Figure 5.9: Shows the greedy controller and how it selects the robot closest to the green line in different states of the simulation

The greedy controller sometimes looses robots and in run 4 it lost 2 robots during 8 minutes and 52 seconds. Since the algorithm always chooses the robot closest to the green line as its next to guide, it has no possibility to save other robots about to leave the arena. As such, the number of robots lost depends on chance. As seen in Figure 5.10 the aerial robot ignores the three robots in danger of exiting the arena, and selects the robot closest to the green line. After it has guided the robot over the green line, it has lost two robots which illustrates how it ignores the bigger picture during its robot selection.

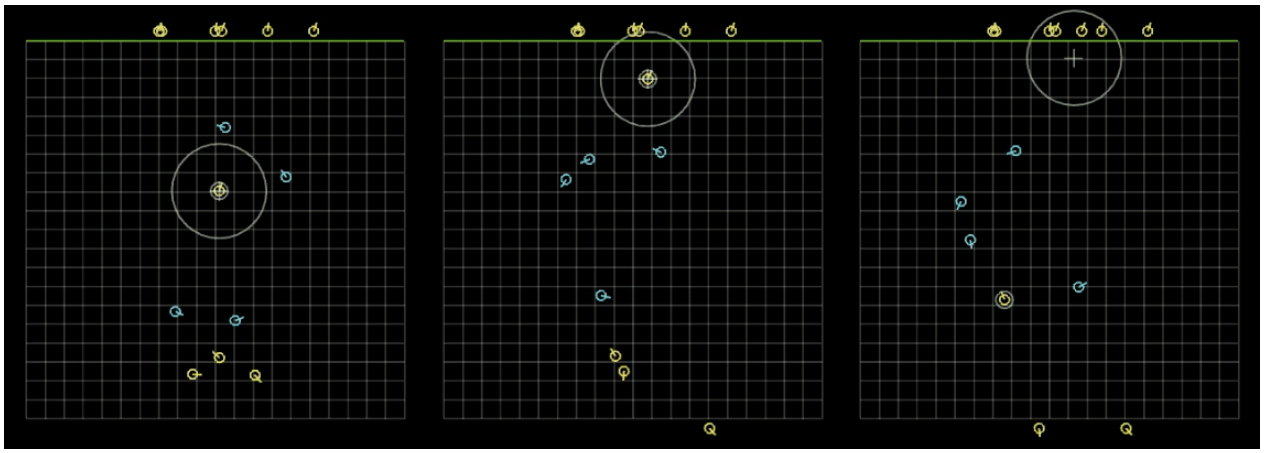

Figure 5.10: Shows how the greedy robots selects and continue to work on the robot closest to the green line, ending with 2 robots exiting the mission arena

5.2.2 Behaviour of Time-Dependent Orienteering Problem with Time Windows Controller

Run	Min. requirement	All requirements	Lost robots	1. robot	2. robot	3. robot	4. robot	5. robot	6. robot	7. robot	8. robot	9. robot	10. robot
1	378	451	0	39	180	219	300	314	331	378	414	420	451
2	374	-	1	37	93	140*	277	300	352	354	374	453	469
3	314	450	0	40	75	80	134	135	179	314	360	412	450
4	556	-	1	189	196	231*	260	358	377	520	556	570	619
5	496	-	1	38	39	219*	309	398	433	448	496	498	511
6	258	351	0	99	119	135	137	196	200	258	271	294	351
7	375	489	0	79	227	255	293	298	351	375	379	471	489
8	316	450	0	78	133	216	274	276	310	316	376	395	450
9	394	-	1	37	196	218	258	291	319*	338	394	488	610
10	357	539	0	51	254	278	279	286	350	357	416	519	539
Avg	381,80	455,00	0,44	68,70	151,20	200,14	252,10	285,20	320,33	380,20	403,60	452,00	493,90

Table 5.17: TDOPTW with ACO controller, where the asterisk denotes that the robot left the arena erroneously

The results in Table 5.17 show that the TDOPTW controller with ACO achieves the minimum requirement of the mission, i.e. guiding 7 robots over the green line in 10 minutes, in all runs. Additionally, in all the runs performed without losing a robot it managed to guide all the robots over the green line within 10 minutes. In two of the instances where it lost a robot it was due to collisions close to the edge of the arena, too far away for the aerial robot to reach in time. Another loss happened because delays in the turn actions resulted in the robot leaving the arena, as shown in Figure 5.14 and further explained below. The last loss happened because a competing plan received a higher score than saving the robot would, although it was expected to leave the arena.

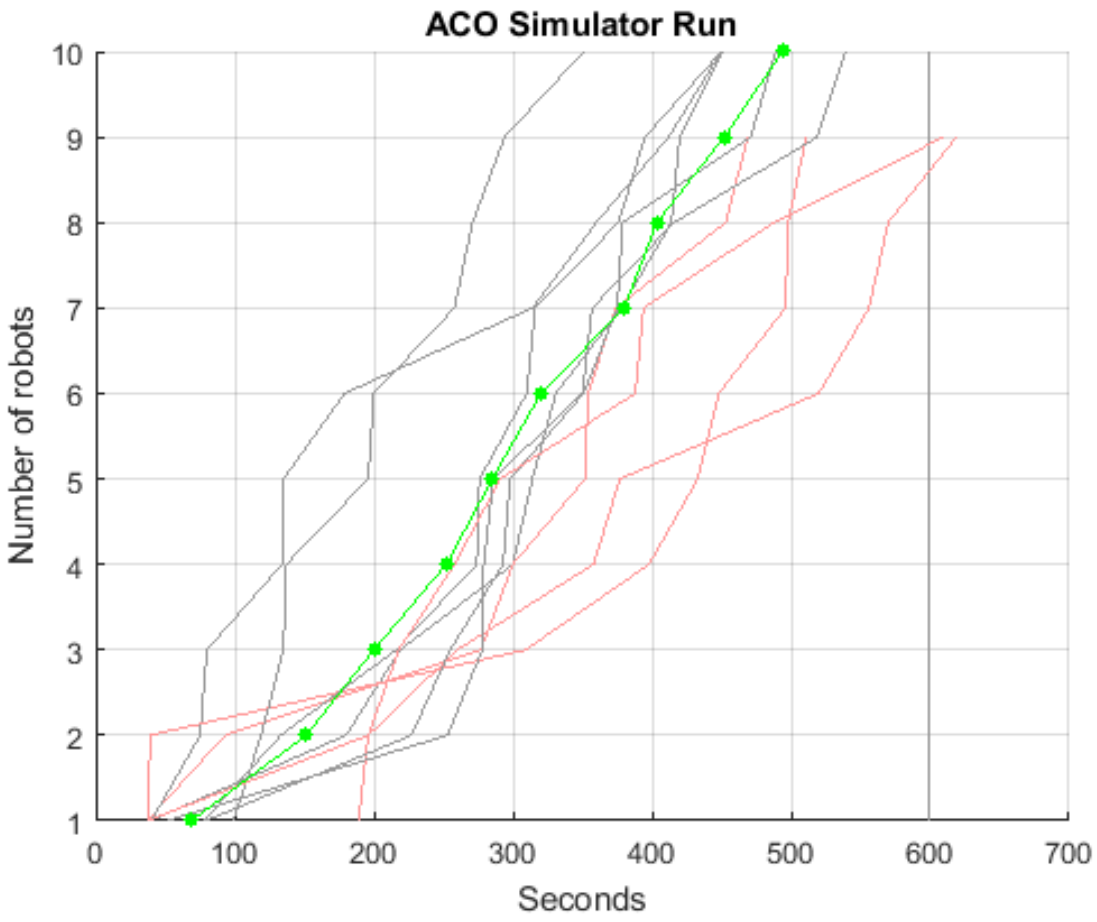

Figure 5.11: Compares the results from each run of the TDOPTW controller using ACO. The green graph is the average, and red graphs show runs where a robot were lost

Figure 5.11 shows that there are great differences between the individual runs. One common trait is that between plateaus the graph leaps multiple increments in a short duration, which testifies to the aerial robot's behavior of guiding multiple robots simultaneously towards the green line, spending more time on each group of robots than greedy does on a single robot. One advantage of TDOPTW is the ability to guide robots about to leave the arena or in dangerous zones towards safe zones, but as observed in the results sometimes robots will escape the arena early in the run, when the collision frequency is high. The interactions from the aerial robot may in some instances increase the collision rate, because more robots are gathered in a smaller area, which leads to greater uncertainty with respect to future states. Although the average value indicates expected performance of the algo-

rithm, it does not reflect any single run very well.

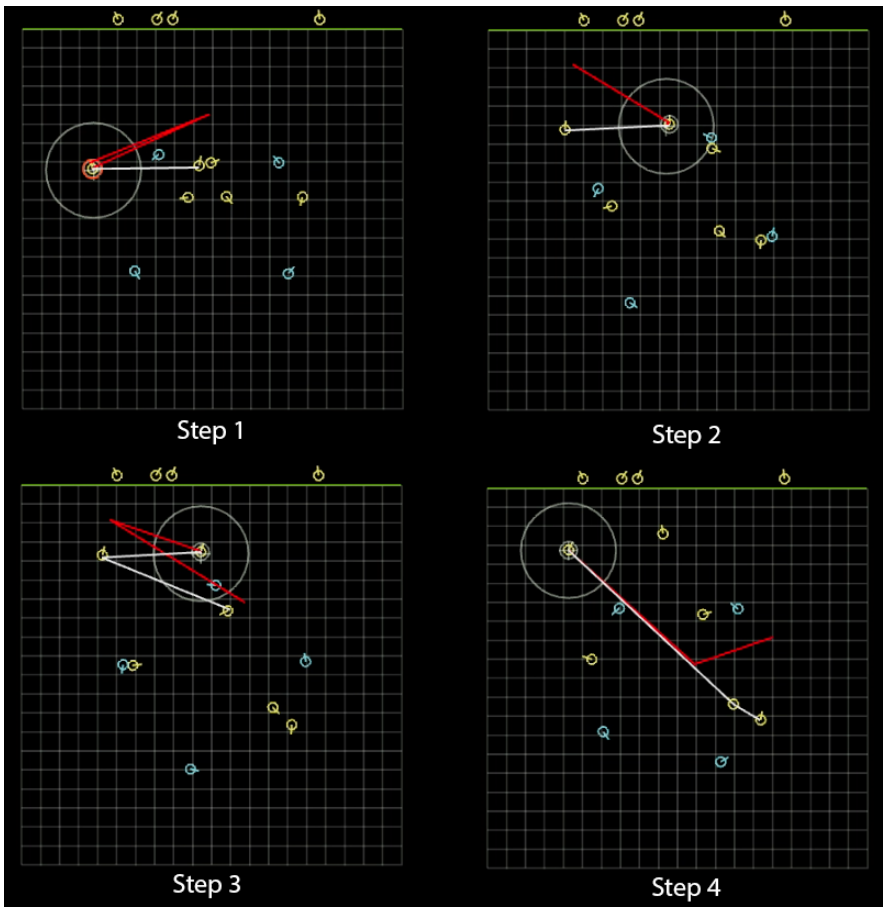

Figure 5.12: A picture series showing the behavior of waiting and handling multiple robots. These pictures are taken form run number 6

Figure 5.12 shows the behavior of guiding multiple robots and waiting for time windows. In step 1 the aerial robot has a plan involving two robots, in step two it waits until step 3 with handling the robot it currently hovers over, before returning to the first robot in step 4. Notice that after each interaction a new plan including the recently handled robot is created: This way it is able to continue working on these two robots and guide them over the green line before handling other robots, behavior made possible by frequent replanning, limited cost budget, and the loose commitment strategy driven by reward comparison. When these two robots are expected to pass the green line unattended, a plan involving other robots is created.

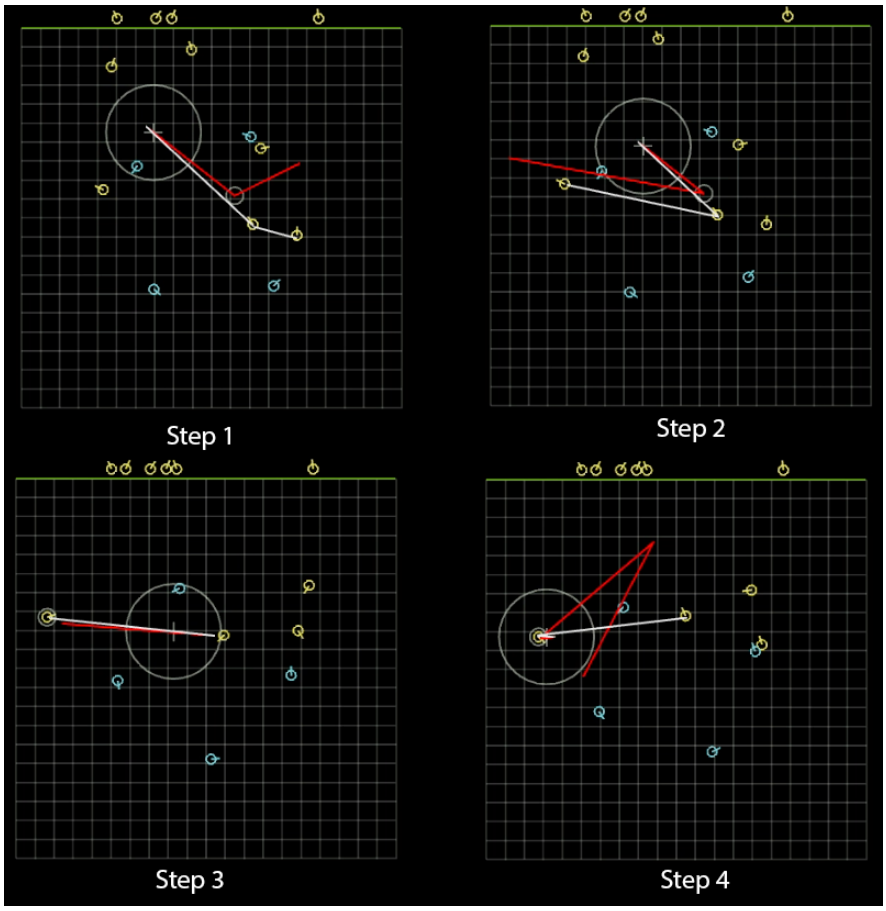

Figure 5.13: A picture series showing the algorithm changing plans to handle a robot in a dangerous area, and successfully saving the robot. These pictures are taken from run number 6

Figure 5.13 shows a situation in step 1 where the aerial robot ignores the two robots expected to pass the green line unattended, and creates a plan involving two other robots. In step 2 a robot on the left side recently passed into what is considered a more dangerous area and the node representation was assigned a higher reward, such that a new plan involving this robot was created. Afterwards the aerial robot continues to handle the two robots from the plan in step 2.

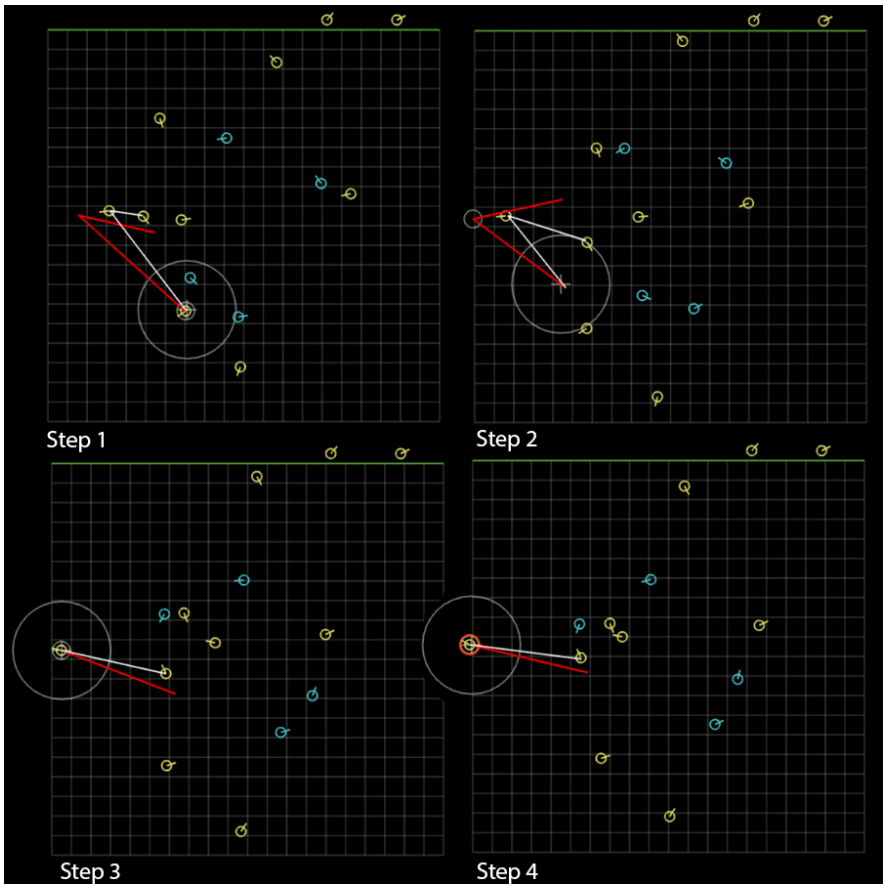

Figure 5.14: A picture series showing an failed attempt of saving a robot due to the action selection. These pictures are taken from run number 4

Figure 5.14 shows one of the instances where a robot left the arena erroneously. As the aerial robot is handling the robot, it tries to turn the robot towards the green line with the fewest number of actions, i.e. two 45 degree turns. The delay between each 45 degree turn enables the robot to progress between the actions resulting in it leaving the bounds of the arena, which could have been prevented by calculating a different action plan.

5.2.3 Controller Comparison

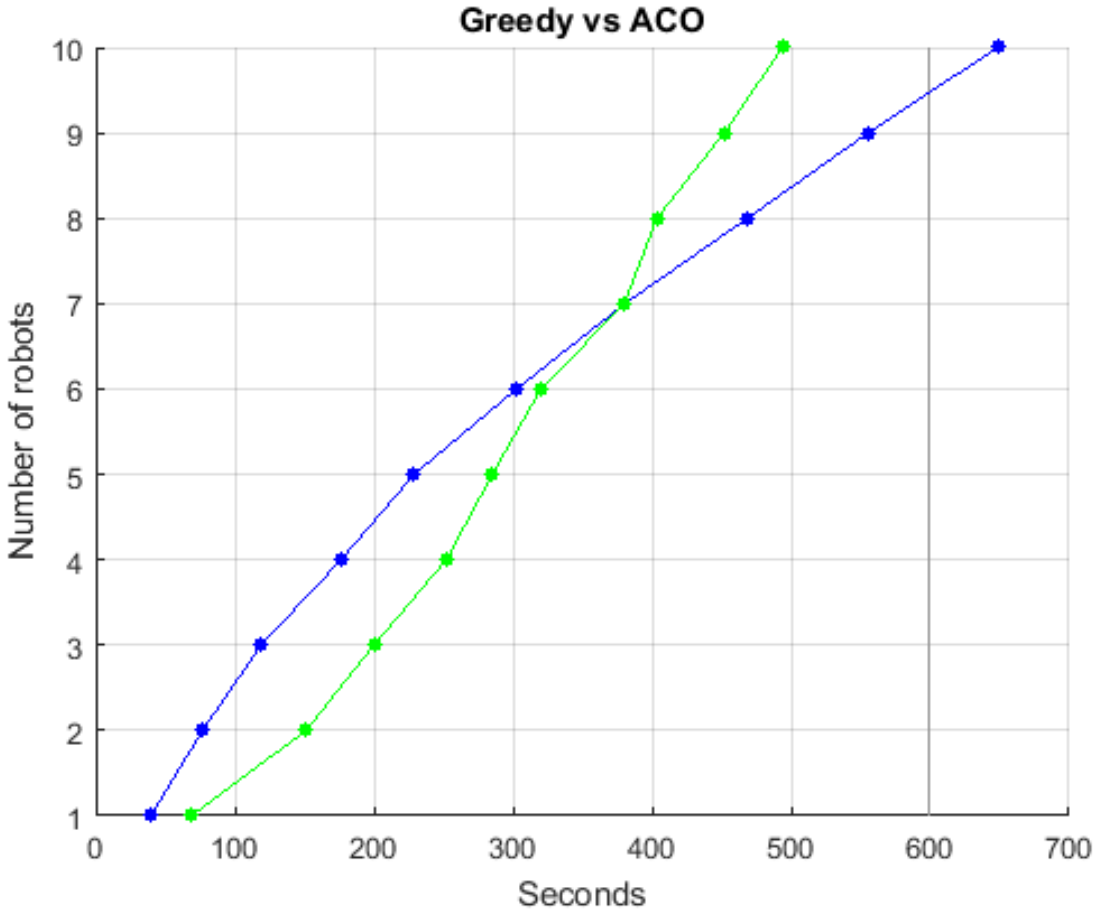

Figure 5.15: Compares the average of the greedy controller, represented by the blue line, and the TDOPTW controller with ACO, represented by the green line

As Figure 5.15 shows, the average completion time of the TDOPTW controller is lower than the greedy controller by about 150 seconds. The graphs intersect at the 7th robot, which happens to be the minimum requirement to complete the mission, as mentioned in Section 2.1.1: The Mission. The most important thing to notice is that the TDOPTW controller spends more time than the greedy controller guiding the first few robots across the green line, but is redeemed when more robots are turned the correct direction and there are fewer robots to handle.

5.3 Summary

IWDA and ACO were tested on OPTW because of the lack of comparable test results for TDOPTW, and was chosen because the only difference is the time-dependency. The test results showed that both IWDA and ACO were able to solve the problem, and although they were far from finding the best known solutions, the range of found solutions showed they were both able to find high quality solutions.

When testing on TDOPTW both algorithms were shown to produce valid solutions, and were shown to find solutions of higher quality than the worst solutions encountered. In 4 out of 6 problem instances ACO outperformed IWDA on most accounts, and generally had a higher average reward and lower reward deviation. Given the test results ACO showed greater promise for use in the simulator than IWDA. Furthermore IWDA was shown to perform better with lower exploitation pressure towards previous solutions, and ACO to have no significant change in performance by using 2-opt local search, except for higher computational cost.

The greedy controller solved the minimum requirement of the mission, guiding 7 robots across the green line in less than 10 minutes, in all instances, but were never able to guide all 10 robots across the green line in less than 10 minutes in any of the runs.

As expected, given the test results on TDOPTW, using IWDA in the TDOPTW controller had worse results than using ACO. It was shown to have problems with committing to a path, and given the poor results testing with IWDA was ended after 3 runs. The TDOPTW controller with ACO showed great promise, and on average solved the minimum requirement of the mission in the same time as the greedy controller. Furthermore, it showed greater ability than the greedy controller to guide robots when fewer robots remained in the state, and in 6 of the 10 instances it guided 10 of the robots over the green line in under 10 minutes, averaging about 150 seconds lower than the greedy controller. Still, its interaction with multiple robots would in some instances induce more frequent collisions and thus greater uncertainties in the state.

Chapter 6

Conclusion

The goal of this research was to implement a scheduling system, modeled as a Time-Dependent Orienteering Problem with Time Windows, for an autonomous aerial robot that guides robots in a dynamic and stochastic environment, for use in the International Aerial Robotics Competition 2016. The main challenges addressed in this work was how to model the environment, what the desired behavior of the aerial robot was, and how to derive the behavior using real-time information.

After abstracting the problem it was compared with known problems, and a good fit was found in the combinatorial problem Time-Dependent Orienteering Problem with Time-Windows. Given the vast amount of possible solutions in even simple Time-Dependent Orienteering Problem with Time Windows instances, the swarm algorithms Intelligent Water Drops Algorithm and Ant Colony Optimization Algorithm were used to perform a heuristic search through possible solutions. During implementation multiple variants and techniques for Intelligent Water Drops Algorithm and Ant Colony Optimization Algorithm were examined, discussed, and tested regarding beneficial properties for the mission environment. Properties of the mission environment not handled in Time-Dependent Orienteering Problem with Time Windows were solved by frequently planning with a short time horizon.

Research Question 1

Which of the algorithms Ant Colony Optimization and Intelligent Water Drops can solve the Time-Dependent Orienteering Problem with Time Windows?

The results in Section 5.1.2: Time-Dependent Orienteering Problem with Time Windows shows that both Ant Colony Optimization Algorithm and Intelligent Water Drops Algorithm are able to solve the optimization problem Time-Dependent Orienteering Problem with Time Windows with adequate results and low computational cost. Since the test instances used have not been tested earlier in the literature it is difficult to ascertain the performance of the algorithms on Time-Dependent Orienteering Problem with Time Windows, but results from solving the Orienteering Problem with Time Windows indicates that other algorithms (or other abbreviations of tested algorithms) could perform better on the Time-Dependent Orienteering Problem with Time Windows.

Research Question 2

Which of the successful algorithms provides the best trade-off between solution quality and computation time?

As shown in Section 5.1.2: Time-Dependent Orienteering Problem with Time Windows Intelligent Water Drops Algorithm and Ant Colony Optimization Algorithm performed best with respect to received reward on different instances, but generally Ant Colony Optimization Algorithm was shown to solve the problems with greater average reward and lower deviation in reward. The Intelligent Water Drops Algorithm was required to use more agents and iterations to obtain comparable results to Ant Colony Optimization Algorithm, which resulted in significantly higher computational cost. Thus the Ant Colony Optimization Algorithm provides the best trade-off between solution quality and computation time.

In context of the simulator, the higher average and lower deviation in reward with Ant Colony Optimization Algorithm was prevalent, as the aerial robot was more consistent regarding commitments when planning, opposed to the time wasted changing plans when Intelligent Water Drops Algorithm was applied.

Research Question 3

Is solving the IARC competition mission 7a as a Time-Dependent Orienteering Problem with Time Windows with the algorithms presented in RQ1 better than a greedy algorithm?

In Section 5.2: Simulator the Time-Dependent Orienteering Problem with Time Windows controller with Ant Colony Optimization Algorithm was shown to perform better than the simple 'greedy controller', while the Intelligent Water Drops Algorithm had problems when applied to the controller, and the results are not able to support any conclusion regarding its performance versus the greedy controller. With the Ant Colony Optimization Algorithm, the Time-Dependent Orienteering Problem with Time Windows controller showed desirable behaviors and were able to guide multiple robots simultaneously, and were in multiple instances able to guide all 10 robots across the green line in under 10 minutes, while the greedy controller only managed to do this once.

However, in the early states of the mission the greedy controller was able to guide robots over the green line faster, and in some instances the Time-Dependent Orienteering Problem with Time Windows controller induced more collisions in the state which led to higher uncertainties and sometimes lost robots. Given that the greedy controller were able to achieve the minimum mission requirement in all the performed runs, the results show the greedy controller should be sufficient to solve the mission, and thus the cost of implementing a Time-Dependent Orienteering Problem with Time Windows controller must be considered before justifying the benefits.

6.1 Research Value

In this research it has been shown that Intelligent Water Drops Algorithm and Ant Colony Optimization Algorithm is able to solve the combinatorial problem Time-Dependent Orienteering Problem with Time Windows, which, to the extent of the authors' knowledge, has not been done before. This also applies to the Orienteering Problem and Orienteering Problem with Time Windows for the Intelligent Water Drops Algorithm.

Furthermore, the research has shown that solving mission 7a of the International Aerial Robotics Competition with Time-Dependent Orienteering Problem with Time Windows as model outperforms an example of a simple greedy algorithm. Both benefits and drawbacks of the Time-Dependent Orienteering Problem with Time Windows controller and the greedy controller have been revealed, which is of great value for Ascend NTNU, and possibly other participants in the competition. As Time-Dependent Orienteering Problem with Time Windows models the IARC mission 7a, this dissertation has contributed to identify a practical application of the model.

6.2 Future Work

Reviewing the simulator results, many of the errors and deficiencies of the system can be prevented by further developing different functionalities of the controller. One of the highlighted incidents where a robot leaves the arena erroneously could have been prevented if the action selection had turned the robot away from danger instead of directly towards the green line. This procedure would incur at least one additional action, but the benefits outweighs the alternative of losing the robot. Additional enhancements of the action selection includes creating action plans directing robots away from collisions and dangerous areas of the arena, and possibly reduce distance between sets of robots that are handled simultaneously.

Further controller enhancements include improving the commitment strategy, which could improve the performance with both algorithms, but especially Intelligent Water Drops Algorithm given the previously discussed commitment issues. The benefits of an improved commitment strategy versus the current solution needs to be considered though, as the issue is complicated and could have been a dissertation in itself.

As seen by the results, the greedy controller excels in the early parts of the mission, and is not susceptible to include additional uncertainties in the state. The Time-Dependent Orienteering Problem with Time Windows controller on the other hand excels at the later states, and as such making the Time-Dependent Orienteering Problem with Time Windows

controller adapt some of the behavior of the greedy controller for the initial four robots would probably be beneficial.

As discussed in Section 4.1: Representing Time-Dependent Cost, the edges of the graph are reused for trail parameters, independent of the dynamic cost and how the edge is used in combination with other edges. This principle works for a brute force search, but goes against the emergent properties of swarm intelligence, and the alternative approach of expanding the graph to include all possible edges, with respect to their costs, should be further researched to determine the difference in performance it can offer, and the computational cost it entails.

The final, and possibly most important, question remains: How will the system perform in the real world? The implementation of the system requires a large amount of information about the environment it operates in. Perception will probably be limited, and thus a model of the parts of the environment not visible needs to be maintained. Uncertainties regarding the position and trajectory of the robots can lead to problems when updating the internal model, e.g. when a robot is found in an expected position but with wrong trajectory, it is uncertain whether the robot has collided or if it is another robot. This problem may be further emphasized if robots leave the arena without the aerial robot perceiving or expecting this. Greater uncertainties in both travel cost and service time may lead to plans failing unexpectedly, as real costs may be higher or lower than expected.

Thus the plans produced by the current implementation are expected to be less robust in the real world, but reducing the available cost budget, changing the commitment strategy, and modifying the action plan to cluster robots within the aerial robot's perception range are modifications worth exploring, which may enable at least parts of the behaviors highlighted in this research.

Given that the greedy controller is easier to implement and the test results show that it satisfies the minimum requirements of the mission, Ascend NTNU should test the greedy controller in the real world and evaluate its performance before considering implementing the Time-Dependent Orienteering Problem with Time Windows controller.

Appendix A

Appendix

A.1 Complete Formulation of Intelligent Water Drops Algorithm Equations

In this section all the equations of the IWDA are showed and explained.

m: Number of IWDs

$s(k) \mid k = 1, \ldots, m$: The solution of water drop k

s_{IB}: Iteration best solution

$vel(k, t)$: The velocity of a water drop k at time t

$soil(k)$: Soil carried by a water drop k

$soil(i, j)$: Soil retained by an edge

```
Input: P, and parameters
InitializeEdgeSoil()
while(not algorithm termination condition):
  InitializeWaterDrops()
  while(not construction termination condition):
    for(k = 1,...,m):
      EdgeSelection()
      UpdateWaterDrop()
      if(f(s(k)) < f(s_IB)):
        s_IB = s(k)
  UpdateEdges(s_IB)
  if(f(s_IB) < f(s*)):
    s* = s_IB
return s*
```

InitializeEdgeSoil

Initialize each edge with soil.

InitializeWaterDrops

On each iteration of the algorithm the water drops are initialized with a predetermined velocity and amount of soil, as well as being scattered, i.e. adding the first component to their solution.

EdgeSelection

This method determines how a water drop k residing in node i chooses its next node j through the selection of an edge component v(i, j). The probability for k to choose v(i, j) is determined by

$$p(k,i,j) = \frac{f(soil(i,j))}{\sum_{\forall l \notin s(k)} f(soil(i,l))}, \tag{A.1}$$

which uses a heuristic function, i.e. measure of how good the solution is, defined by

$$f(soil(i,j)) = \frac{1}{\varepsilon + g(soil(i,j))} \tag{A.2}$$

where ε is a small positive number used to prevent division by zero, and

$$g(soil(i,j)) = \begin{cases} soil(i,j) & if \min\limits_{\forall l \notin s(k)} \geq 0 \\ soil(i,l) - \min\limits_{\forall l \notin s(k)} (soil(i,l)) & otherwise. \end{cases} \tag{A.3}$$

such that an edge with little soil is chosen more often. After the edge is selected, it is added to the solution of water drop k.

UpdateWaterDrop

Given the static velocity update parameters (a_v, b_v, c_v), every time a water drop moves between nodes, its velocity is updated by

$$vel(k, t+1) = vel(k,t) + \frac{a_v}{b_v + c_v \times soil(i,j)}. \tag{A.4}$$

The amout of soil carried by the water drop k is updated according to

$$soil(k) = soil(k) + \Delta soil(i,j) \tag{A.5}$$

and the amount of soil left in the edge after the water drop has traversed it is

$$soil(i,j) = (1 - \rho_n) \times soil(i,j) - \rho_n \times \Delta soil(i,j) \tag{A.6}$$

where $0 < \rho_n < 0$. Given the static soil update parameters (a_s, b_s, c_s), the amount of soil

removed from the path and carried by the water drop is defined by

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \times time(i,j \mid vel(k,t+1))}, \tag{A.7}$$

where the function *time* describes the duration of traversing the edge as a function of the water drops velocity, defined by

$$time(i,j \mid vel(k,t+1)) = \frac{HUD(i,j)}{vel(k,t+1)} \tag{A.8}$$

where HUD(i, j) is a desirability heuristic for v(i, j).

UpdateEdges

After all water drops have constructed a tour, the edges of the best solution found in the current iteration is updated:

$$soil(i,j) = (1+\rho_{IWD}) \times soil(i,j) - \rho_{IWD} \times soil(s_{IB}) \times \frac{1}{m-1} \tag{A.9}$$

where ρ_{IWD} is a positive constant.

Termination Conditions

Any termination condition may be specified, both for the algorithm and solution construction. A common condition is to run a fixed, finite number of iterations.

A.2 International Aerial Robotics Competition

A.2.1 Previous Missions

Previously six different challenges has been completed, each taking one to eight years to accomplish. The next sections will give a brief description of each mission.

A.2.1.1 First Mission

The initial mission was for the aerial robot to move a metallic disk from one side of an arena to the other, completely autonomously. It took the competitors two years to accomplish autonomous take-off, flight, and landing, and another two years to complete the mission in 1995.

A.2.1.2 Second Mission

The second mission consisted of searching for toxic waste, as signified by partially buried waste drums, and identifying the contents by hazard labels found on the drums. The mission was finished in 1997.

A.2.1.3 Third Mission

The third mission was a search and rescue mission, requiring the robots to search for survivors and dead amid fires, broken water mains, toxic gas, and rubble, and report the information back to a rescue team. The mission was completed in 2000.

A.2.1.4 Fourth Mission

The fourth mission consisted of three similar scenarios:

- Hostage rescue mission, where the robot had to fly 3 km, identify the embassy building the hostages were being held, and in some way relay pictures of the hostages back to the station.
- A tapestry was reported by an archaeological team in a mausoleum a short time before their death, due to an ancient virus, where the robots task is to find the tapestry and relay images back to base before the destruction of the mausoleum.
- After an explosion at a nuclear facility, only one reactor remain intact. The robots mission is to relay pictures of the instrument panel inside the nuclear facility in order for a team at the base to determine if a melt-down is imminent.

All requirements of the challenge, except completing in under 15 minutes, was accomplished in 2008. As the organizers of the challenge deemed it an insignificant challenge to accomplish the time requirement, the mission was terminated as successful.

A.2.1.5 Fifth Mission

The fifth mission was an extension of the fourth, where the task was to navigate a complex interior of a building without the aid of global-positioning navigational aids, in order to reach a designated target and relay images back to base. The mission was accomplished in 2009.

A.2.1.6 Sixth Mission

Again, the next mission was an extension of the previous one, where the goal was to navigate an unknown structure and steal a flash drive, replacing it with an identical flash drive to avoid the detection of theft. The mission was completed in 2013.

A.2.2 Scoring

Participants receive points in two categories: Effectiveness measures, and subjective measures. The effectiveness measures consists of rewards and penalties according to the skill displayed in completing the mission in the following categories:

- Rewards
    - Ground robot crossing green boundary, having been touched on top
    - Autonomous operation
    - Obstacle avoidance
- Penalties
    - Ground robot not crossing green boundary, or crossing it without being touched on top
    - Per minute penalty until completion

The subjective measure consists of rewards in the following categories, with predefined criteria:

- Elegance of design and craftsmanship
- Innovation in air vehicle design
- Safety of design to bystanders
- Journal paper
- Best team T-shirt

A.2.3 Competition Venues

From 2012 the competition has been served in two venues: In USA, mainly serving American, European, and African teams; and in China mainly serving Asian and Oceanic teams.

Despite this general distribution, teams may choose which venue to participate in. As this paper is written, the specific locations of the 2016 challenge are yet to be decided.

A.3 Additional Benchmark Results

Unless stated otherwise, all results below are gathered from 100 runs on each problem instance.

	Without local search								With local search							
Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Max Time	Avg Time	Min Time	Dev time	Max Reward	Avg Reward	Min Reward	Dev Reward	Max Time	Avg Time	Min Time	Dev time
c101	130.00	130.00	130.00	0.00	0.0831	0.0721	0.0654	0.0039	130.00	130.00	130.00	0.00	0.0929	0.0797	0.0732	0.0040
c109	210.00	204.00	200.00	4.90	0.1304	0.1168	0.1064	0.0056	210.00	203.40	200.00	4.74	0.1482	0.1352	0.1249	0.0055
r107	155.00	149.13	148.00	1.79	0.1285	0.1150	0.1060	0.0052	150.00	149.07	149.00	0.26	0.1556	0.1412	0.1280	0.0058
r109	154.00	146.97	138.00	6.14	0.1283	0.1066	0.0979	0.0054	153.00	148.03	140.00	5.91	0.1822	0.1344	0.1187	0.0103
rc101	150.00	148.70	140.00	3.36	0.0876	0.0759	0.0693	0.0038	150.00	150.00	150.00	0.00	0.1041	0.0932	0.0833	0.0045
rc106	150.00	150.00	150.00	0.00	0.0933	0.0842	0.0753	0.0042	150.00	148.90	140.00	3.13	0.1106	0.0980	0.0895	0.0047

Table A.1: Comparing ACO with 2-opt local search and without local search

	High exploitation								Low exploitation							
Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Max Itr	Avg Itr	Min Itr	Dev Itr	Max Reward	Avg Reward	Min Reward	Dev Reward	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	170.00	112.60	100.00	12.54	92.00	10.00	1.00	14.30	160.00	132.90	110.00	8.87	99.00	42.00	1.00	27.81
c109	180.00	138.30	110.00	14.29	97.00	24.00	1.00	26.34	180.00	150.70	110.00	15.31	100.00	44.00	2.00	30.59
r107	142.00	111.95	92.00	11.08	99.00	8.00	1.00	16.44	145.00	120.21	97.00	9.62	100.00	35.00	1.00	28.01
r109	142.00	111.32	78.00	12.18	100.00	11.00	1.00	17.75	143.00	119.24	102.00	9.92	99.00	20.00	1.00	21.36
rc101	150.00	115.40	80.00	14.03	13.00	2.00	1.00	1.93	160.00	136.10	100.00	11.04	97.00	25.00	1.00	26.84
rc106	140.00	106.20	70.00	13.40	24.00	3.00	1.00	3.82	150.00	123.80	100.00	10.66	93.00	28.00	1.00	22.03

Table A.2: Comparing IWD with high and low exploitation

Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Max Cost	Avg Cost	Min Cost	Dev cost	Max Time	Avg Time	Min Time	Dev time	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	130.00	130.00	130.00	0.00	1,078.65	900.68	858.93	86.19	0.0661	0.0531	0.0511	0.0019	1.00	1.00	1.00	0.00
c109	210.00	205.40	200.00	4.98	1,216.64	1,206.56	1,196.54	9.91	0.0913	0.0870	0.0838	0.0018	10.00	3.00	1.00	2.72
r107	152.00	149.36	149.00	0.97	226.22	219.79	218.92	2.37	0.1313	0.0904	0.0825	0.0096	4.00	2.00	1.00	1.26
r109	153.00	148.21	139.00	6.16	222.36	212.08	207.71	5.04	0.1072	0.0875	0.0780	0.0063	8.00	3.00	1.00	2.35
rc101	150.00	150.00	150.00	0.00	236.90	230.21	228.32	3.56	0.0708	0.0583	0.0537	0.0038	7.00	3.00	1.00	2.29
rc106	160.00	148.40	140.00	6.74	235.64	223.61	219.72	5.50	0.0661	0.0614	0.0586	0.0014	7.00	3.00	1.00	2.24

Table A.3: ACO on TDOPTW problem instances using rank selection

Name	Max Reward	Avg Reward	Min Reward	Dev Reward	Max Cost	Avg Cost	Min Cost	Dev cost	Max Time	Avg Time	Min Time	Dev time	Max Itr	Avg Itr	Min Itr	Dev Itr
c101	130.00	130.00	130.00	0.00	858.93	858.93	858.93	0.00	0.0613	0.0533	0.0513	0.0015	1.00	1.00	1.00	0.00
c109	210.00	209.90	200.00	0.99	1,216.64	1,211.28	1,169.78	14.73	0.0975	0.0879	0.0851	0.0020	7.00	3.00	1.00	2.22
r107	154.00	148.95	148.00	1.83	218.92	212.39	208.22	5.12	0.0963	0.0865	0.0832	0.0021	7.00	2.00	1.00	2.16
r109	153.00	149.02	140.00	5.35	225.38	211.73	207.71	6.13	0.0879	0.0806	0.0767	0.0018	9.00	5.00	4.00	1.91
rc101	150.00	150.00	150.00	0.00	236.90	232.44	228.32	4.29	0.0632	0.0565	0.0544	0.0013	4.00	1.00	1.00	1.46
rc106	150.00	148.50	140.00	3.57	226.10	222.63	219.72	2.31	0.0678	0.0622	0.0601	0.0013	9.00	2.00	1.00	2.54

Table A.4: ACO on TDOPTW problem instances using tournament selection

Bibliography

Alijla, B. O., Wong, L.-P., Lim, C. P., Khader, A. T., and Al-Betar, M. A. (2014). A modified intelligent water drops algorithm and its application to optimization problems.

Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. (2007). The traveling salesman problem.

Beni, G. and Wang, J. (1989). Swarm intelligence in cellular robotic systems. *NATO ASI Series*, 102:703–712.

Bin, Y., Zhong-Zhen, Y., and Baozhen, Y. (2008). An improved ant colony optimization for vehicle routing problem. *European Journal of Operational Research*, 196:171–176.

Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research 88*, pages 475–489.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT Press, 3 edition.

C.Verbeeck, Sörensen, K., Aghezzaf, E.-H., and Vansteenwegen, P. (2013). A fast solution method for the time-dependent orienteering problem. *European Journal of Operational Research*, 236:419–432.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Dorigo, M., Birattari, M., and Stützle, T. (2006). Ant colony optimization: Artificial ants as a computational intelligence technique. *IEEE Computational Intelligence Magazine*.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344:243–278.

Dorigo, M., Colorni, A., and Maniezzo, V. (1991). Distributed optimization by ant colonies. *Proceedings of Ecal91*, pages 134–142.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1.

Floreano, D. and Mattiussi, C. (2008). *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT Press.

Fomin, F. V. and Lingas, A. (2002). Approximation algorithms for time-dependent orienteering. *Information Processing Letters 83*, pages 57–62.

Gambardella, L. M. and Dorigo, M. (1995). Ant-q: A reinforcement learning approach to the traveling salesman problem. *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, pages 252–260.

Garcia, A., Arbelaitz, O., Vansteenwegen, P., Souffriau, W., and Linaza, M. T. (2010). Hybrid approach for the public transportation time-dependent orienteering problem with time windows.

Ghallab, M., Nau, D., and Traverso, P. (2004). Hierarchical task network planning. *Automated Planning*, page 229–261.

Gunawan, A., LAU, H. C., and Lu, K. (2015). The latest best known solutions for the team orienteering problem with time windows (toptw) benchmark instances.

Hespanha, J., Kim, H. J., and Sastry, S. Multiple-agent probabilistic pursuit-evasion games. *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No.99CH36304)*.

http://www.mech.kuleuven.be. The Team Orienteering Problem with Time Windows: Test Instances. http://www.mech.kuleuven.be/en/cib/op. [Online; accessed 26-May-2016].

International Aerial Robotics Competition (2015). Official rules for the international aerial robotics competition. http://www.aerialroboticscompetition.org/downloads/mission7rules_081015.pdf. [Online; accessed 08.12.15].

Kamkar, I., Akbarzadeh-T, M.-R., and Yaghoobi, M. (2010). Intelligent water drops: A new optimization algorithm for solving the vehicle routing problem.

Kantor, M. G. and Rosenwein, M. B. (1992). The orienteering problem with time windows.

Laporte, G. (1991). The traveling salesman problem: An overview of exact and approximate algorithms.

Liang, Y.-C., Kulturel-Konak, S., and Smith, A. E. (2002). Meta heuristics for the orienteering problem.

Mavrovouniotis, M. and Yang, S. (2013). Ant colony optimization with immigrant schemes for the dynamic travelling salesman problem with traffic factors.

Montemanni, R., Weyland, D., and Gambardella, L. (2011). An enhanced ant colony system for the team orienteering problem with time windows. *2011 International Symposium on Computer Science and Society*.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451.

Parpinelli, R. S. and Lopes, H. S. (2011). New inspirations in swarm intelligence: A survey. *International J. Bio-Inspired Computation*, 3.

Reinelt, G. (2008). Symmetrical tsp data sets. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/. [Online; accessed 08-December-2015].

Righini, G. and Salani, M. (2006). Dynamic programming for the orienteering problem with time windows.

RobotShop Inc (2015). iRobot Releases iRobot Create Platform. http://www.robotshop.com/blog/en/hackers-rejoice-irobot-releases-irobot-create-platform-3892. [Online; accessed 08.12.15].

Shah-Hosseini, H. (2007). Problem solving by intelligent water drops.

Shah-Hosseini, H. (2008). Intelligent water drops algorithm: A new optimization method for solving the multiple knapsack problem. *Computing and Cybernetics*, 1(2):193–212.

Strombom, D., Mann, R. P., Wilson, A. M., Hailes, S., Morton, A. J., Sumpter, D. J. T., and King, A. J. (2014). Solving the shepherding problem: heuristics for herding autonomous, interacting agents. *Journal of The Royal Society Interface*, 11(100).

Stültze, T. and Hoos, H. (1996). Improving the ant system: A detailed report on the max-min ant system.

Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *J. OpI Res. Soc.*, 35(9):797–809.

Vansteenwegen, P., Souffriau, W., Berghe, G. V., and Oudheusden, D. V. (2009). Iterated local search for the team orienteering problem with time windows. *Computers and Operations Research 36*, pages 3281–3290.

Vansteenwegen, P., Souffriau, W., and Oudheusden, D. V. (2010). The orienteering problem: A survey.

Yu, B., Yang, Z.-Z., and Yao, B. (2009). An improved ant colony optimization for vehicle routing problem. *European Journal of Operational Research*, 196:171–176.