Dipl.-Ing. Vera Mersheeva

UAV Routing Problem for Area Monitoring in a Disaster Situation

DISSERTATION

submitted in fulfilment of the requirements for the degree of
Doktorin der Technischen Wissenschaften

Alpen-Adria Universität Klagenfurt
Fakultät für Technische Wissenschaften

Mentor and 1st Evaluator
O.Univ.-Prof. Dipl.-Ing. Dr. Gerhard Friedrich
Alpen-Adria Universität Klagenfurt
Institut für Angewandte Informatik

2nd Evaluator
Prof. Dr. rer. nat. Frank Ortmeier
Otto-von-Guericke-Universität Magdeburg
Institut für Verteilte Systeme

Klagenfurt, March/2015

Affidavit

I hereby declare in lieu of an oath that

- the submitted academic paper is entirely my own work and that no auxiliary materials have been used other than those indicated,
- I have fully disclosed all assistance received from third parties during the process of writing the paper, including any significant advice from supervisors,
- any contents taken from the works of third parties or my own works that have been included either literally or in spirit have been appropriately marked and the respective source of the information has been clearly identified with precise bibliographical references (e.g. in footnotes),
- to date, I have not submitted this paper to an examining authority either in Austria or abroad and that
- the digital version of the paper submitted for the purpose of plagiarism assessment is fully consistent with the printed version.

I am aware that a declaration contrary to the facts will have legal consequences.

(Signature) (Place, date)

Abstract

The use of unmanned aerial vehicles (UAVs) is gaining popularity in various areas such as disaster management, agricultural surveillance, urban terrain surveillance, military operations, and construction site monitoring. In particular, the use of micro UAVs, i.e., drones that can be carried by a human, has garnered a lot of attention in research and applications because of their ease of use and low costs.

In this thesis we consider the exploration of a disaster scene, e.g., earthquake or train accident, using a fleet of micro UAVs equipped with cameras. The terrain is represented as a set of locations to be observed with potentially different levels of importance. Obviously, the main intention of the aerial exploration is to always have the most up-to-date information for every point of interest. Therefore, the UAVs have to visit such locations as often as possible but avoid long time spans between revisits. UAVs are typically controlled manually by at least one human operator for each vehicle. In case of search and rescue operations, human resources cannot be wasted on steering the drones. Consequently, automatic planning is required and it has to be done fast, while the fleet of drones is being prepared to start the flight mission.

In order to model the described task with its real-life constraints, we introduced the continuous monitoring problem (CMP) and its extensions with inter-depot routes (CMPID) and priorities (CMPIDP). Continuous monitoring means that a fleet of vehicles has to periodically visit a set of locations. Vehicles are constrained by the capacity of their energy storage, e.g., batteries. The capacity can be renewed at one of the available base stations. The goal is to find a plan where the fleet visits locations uniformly and as often as possible so that long intervals between revisits are avoided.

To solve these problems, we propose several construction approaches: modified Clarke and Wright algorithm, queue-based insertion heuristic, inter-depot insertion heuristic (IDIH), and IDIH with reservations. The solutions obtained by these algorithms are improved by a metaheuristic approach based on the variable neighborhood search. The suggested methods were evaluated on numerous instances including real-life scenarios. The evaluation showed that the developed methods are capable of providing near-optimal solutions in a short time.

Acknowledgements

This work has been initiated and performed mostly within the projects "Collaborative Microdrones" (cDrones) and "Self-organizing Intelligent Network of UAVs" (SINUS). I would like to thank the teams of both projects and the research cluster Lakeside Labs for their support and inspiration.

I would like to acknowledge an appreciation to my supervisor Prof. Friedrich for believing in me and my ideas, for the fruitful discussions and advices that helped me to achieve this important goal. I would also like to thank Prof. Ortmeier for his valuable comments that helped to improve the quality of the dissertation.

I am very grateful to the friends that I feel honored to meet during this time: Ursula Rotter, Anna Rybokon and her family, John NA Brown, Samirah Hayat, and Udaya Lakshmi Cherukuri. The list is not full and can be greatly extended. You have enriched my stay here in many ways.

I would also like to thank Sergey Alatartsev, John NA Brown, Alexander Gogolev, Anna Ryabokon and Konstantin Schekotihin whose suggestions have helped to shape this work and certainly improved the style and the language.

My deepest gratitude goes to my family and Sergey Alatartsev for their love, encouragement and all those numerous great moments that kept me going. Without your support, this dissertation would not have been possible.

Klagenfurt, March 2015 Vera Mersheeva

Contents

List of Figures

List of Tables

List of papers

1. V. Mersheeva and G. Friedrich. Routing of multiple micro UAVs for rescue missions. In *Austrian Robotics Workshop (ARW)*, 2012.

2. V. Mersheeva and G. Friedrich. Routing for continuous monitoring by multiple micro UAVs in disaster scenarios. In *European Conference on Artificial Intelligence (ECAI)*, pages 588–593, 2012.

3. Vera Mersheeva, Daniel Wischounig-Strucl, Markus Quaritsch, Saeed Yahyanejad, Robert Kuschnig, Bernhard Rinner and Gerhard Friedrich. Lightscribing the sky with micro UAVs. In *1st microdrones International Research Workshop*, 2012.

4. V. Mersheeva and G. Friedrich. Continuous monitoring problem for disaster management. In *Meeting EURO working Group on Vehicle Routing and Logistics Optimization (VeRoLog)*, 2013.

5. V. Mersheeva and G. Friedrich. Variable neighborhood search for continuous monitoring problem with inter-depot routes. In I. J. Timm and M. Thimm, editors, *KI 2013: Advances in Artificial Intelligence, volume 8077 of Lecture Notes in Computer Science*, pages 106–117. Springer Berlin Heidelberg, 2013.

6. V. Mersheeva and G. Friedrich. Multi-UAV monitoring with priorities and limited energy resources. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2015

Chapter 1

Introduction

Over the last few decades, we have seen the deployment of micro unmaned aerial vehicles (UAVs or man-portable drones) for a variety of civil applications, from aerial photography to the delivery of medicine in Africa. They are affordable, easy and fast to deploy. These machines can utilize the third travel dimension, thus, avoiding numerous problems with obstacles.

This thesis focuses on the use of UAVs during or after a disaster. Drones do not require direct human presence at the scene and, thus, prevent unnecessary risks to human lives. They can be deployed to monitor the development of a situation, to aid in the search and rescue of survivors, to trace the safest path for a rescue team, and to conduct post-disaster inspections. This technology has already demonstrated its potential in the aftermath of several catastrophes like hurricane Katrina in the USA or the Tohoku earthquake and tsunami at the Fukushima nuclear power plant in Japan.

Let us consider a scenario where a team of rescuers arrives at the location during or after a disaster, e.g., an earthquake or a forest fire. Needless to say that any information about the situation is essential. Very often it cannot be gained from the ground, and aerial surveillance is needed. However, human resources cannot be wasted on steering the drones. Therefore, areal observations must be provided in an automatic way.

One way to obtain such observations is to send a drone to a certain location above the area and record a video or take a picture of the whole disaster scene. In order to cover a large area in one shot, a vehicle has to either reach a certain altitude or carry a wide-angle lens camera. The first option is not possible due to legal regulations, when drones are not permitted to fly above certain altitude due to safety reasons. Carrying a wide-angle lens camera is also infeasible due to limited payload of a micro UAV. The way to resolve this problem is to split the overall area into rectangles and take

pictures of the rectangles from a lower altitude. Then this data is transmitted to the ground station, where received pictures are stitched together into a high-resolution overview image [72]. Locations where photos will be taken are called picture points. Obviously, the main intention of the aerial exploration is to always have the most up-to-date information for every rectangle. Therefore, the UAVs have to visit picture points as often as possible and avoid long time spans between revisits.

In this thesis, multi-copters were used as a platform for the monitoring task. A multi-copter is UAV equipped with several rotors powered by a battery, see Fig. 1.1. Their advantage over the fixed-wing UAVs is the ability to hover at one location in order to take a picture. The down side is their short flight time due to limited battery capacity. It amounts to approximately 10-45 minutes flight time depending on the type of a vehicle, environmental conditions and payload. To extend monitoring time, the UAVs have to visit the ground base stations where charged batteries are stored.

Figure 1.1: Quadcopter AscTec Pelican

Steering a UAV typically requires at least basic theoretical training as well as practical experience. Controlling a flying machine to navigate through the given picture points while taking into account its battery capacity and weather conditions is not a trivial task. This is another reason for the automation of the monitoring process by introducing a route planning algorithm.

The goal of a route planning algorithm is to obtain a set of routes for a fleet of vehicles to consistently survey a set of picture points. A route is a sequence of way points that starts at the previous location of a vehicle and ends at one of the base stations. Vehicles might have different average flight speeds and battery capacities that cannot be exceeded. The optimal solution of such a monitoring problem is a set of routes that maximizes the number of observations, e.g., taken pictures, and

minimizes the time delays between observations for all points over a certain period of time.

Fig. 1.2 illustrates fire-fighters training at a factory[1]. The input for a route planning algorithm is presented on the left side of the figure. The picture points are depicted as circles and the base stations are displayed as red dots. The image on the right is a partial screenshot of the user interface during the exploration. The background picture from Google Earth™ (layer 1) is used to define the area of interest. During the initial flights, a high-resolution overview image is placed on top (layer 2). Afterwards layer 2 contains the latest complete overview image. The updates of the overview image are displayed in higher contrast on layer 3. The black polygonal paths are the routes which the UAVs have followed so far in their latest flights.

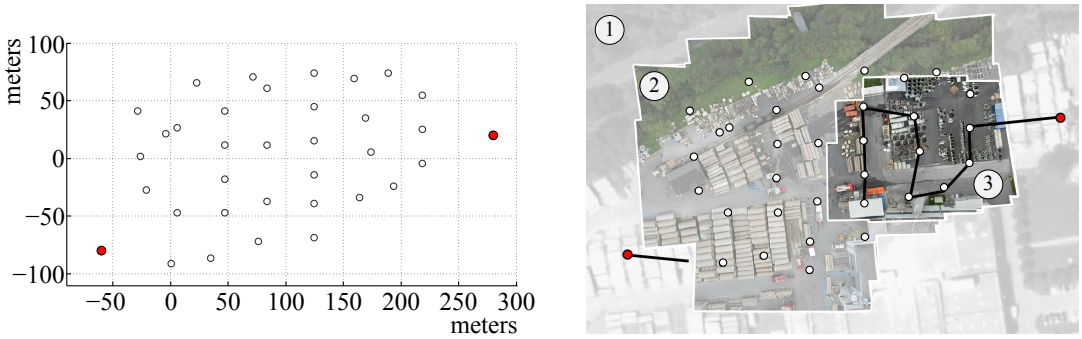


Figure 1.2: Exploration of an area by a fleet of UAVs. Left: input data for a routing algorithm. Right: intermediate results of the exploration.

In rescue missions, different subareas can have different importance. For instance, the most important subareas are territories on the edge of a forest fire or those that most likely have people who need help. These subareas should be visited more often. This is demonstrated in Fig. 1.3 with expected distribution of visits. The larger the number of visits, the larger the circle.

Several existing problems can model consistent surveillance of target points. The operational research (OR) community has considered several related route planning problems, e.g., the periodic vehicle routing problem with intermediate facilities [5] as well as the vehicle routing problem with multiple time windows and multiple visits [31]. In the field of robotics Machado et al. [52] introduced the patrolling task originally applied to ground robots [3], [83] and later to areal vehicles [69], [93]. However, these works do not consider all the main aspects of the monitoring problem

[1]Apart from this training, we have participated in a number of other events and demonstrations. More information about the project and videos of the test flights can be found on `http://uav.lakeside-labs.com/` and `http://youtu.be/SvL8rTUNh1c` (last accessed on March 8, 2015).

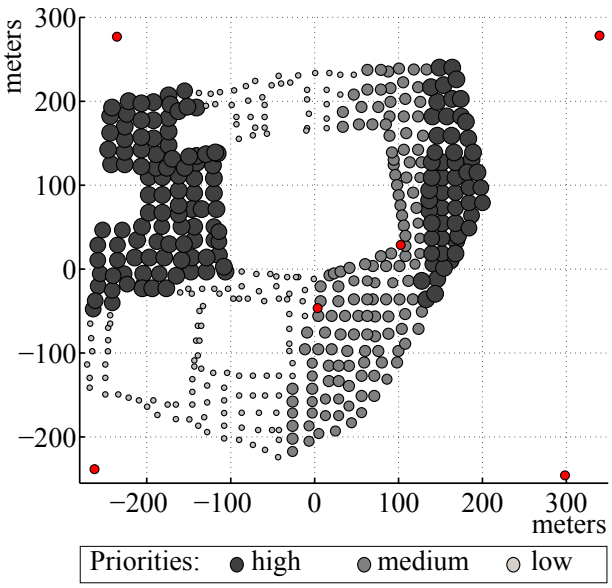


Figure 1.3: Required distribution of number of visits at points with three levels of importance.

for micro UAVs. For example, the patrolling task does not limit battery capacity, and the OR problems typically optimize purchased fuel or the total travel time instead of maximizing gathered information. In order to fill the gap, we introduce the continuous monitoring problem (CMP) and its extensions with inter-depot routes (CMPID) and priorities (CMPIDP). These problems arise in many real-life applications such as aerial continuous surveillance of a potential crime location or a concert.

The goal of this thesis is to propose efficient route planning algorithms for these problems that would be sufficiently fast for rescue operations and able to react to a number of environmental changes.

There are two main groups of approaches for solving route planning problems: exact algorithms and heuristics. Methods from the first group solve problems optimally. However, due to problem complexity, they are limited to small-size scenarios. In contrast to the exact methods, heuristics construct feasible but not necessarily optimal solutions that are sufficient for most applications.

In this thesis we propose several solving methods for the aforementioned problems: four heuristic approaches to construct initial solutions and use a method called variable neighborhood search (VNS) for improving it. Construction heuristics include modified Clarke and Wright algorithm (CW) and queue-based insertion heuristic (QIH) for the CMP, inter-depot insertion heuristic (IDIH) for the CMPID and IDIH with battery reservations (IDIH-Reserve) for the CMPIDP. These heuristics

start with an empty solution and iteratively extend it based on a certain metric like distance and arrival time.

There are two main contributions of this thesis:

1. The thesis proposes definitions of a real-life problem and its extensions that, to the best of our knowledge, have not been studied before. Practical importance of the monitoring problem with capacity constraints cannot be underestimated, as it has a wide range of applications, e.g., consistent aerial surveillance of a disaster area, a crime scene or a concert.

2. The thesis describes solution methods for the proposed monitoring problems. The developed heuristics yield good results during the evaluation. They obtain near-optimal solutions on instances, where an optimum can be computed. Real-life scenarios are planned by construction heuristics in a short time, e.g., less than 11 seconds for the largest scenario. Thus, the methods can be used for time-critical missions. The insertion heuristics are flexible and can be adapted for the new constraints with minor modifications. Finally, solution quality can be improved further with variable neighborhood search, e.g., by up to 30% in 10 minutes computational time.

The work described in this thesis has already been reported in several publications. Solving techniques for the CMP have been published in [55] and [54]. Approaches for the CMPID and CMPIDP have been presented in [57] and [58], respectively.

The thesis is organized as follows. Chapter 2 provides formal definitions of the continuous monitoring problem and its extensions. Chapter 3 starts with a description of related state-of-the-art methods and proposed heuristics and then provides an explanation of the constraint-based programming approach used to obtain optimal solutions. The chapter concludes by explaining application of the proposed methods to problems with dynamic environments. Chapter 4 reports results achieved by the proposed heuristics in a number of conducted studies. They include tuning of the methods parameters, comparison with optimum on smaller instances, and performance evaluation on large scenarios.

Chapter 2

Monitoring Problems

Let us briefly explain the routing problems and their differences. The vehicle routing problem (VRP) [89] is one of the most well-studied route planning problems. It is stated as follows: a fleet of identical vehicles located at a depot has to visit a set of customers to deliver demanded amount of goods. Each vehicle has a limited carrying capacity for the goods. The goal is to calculate a route for every vehicle so that it visits every customer exactly once with minimal total travel time.

The continuous monitoring problem (CMP) introduced in this thesis is a variation of the VRP. It deals with periodic visitation of points instead of single visits at each point. Vehicles used for monitoring have limited travel time defined by energy capacity. The travel time includes the time required to travel between the points as well as the time to service each point, e.g., to take picture at a picture point. A vehicle can renew its energy capacity at a home station: an assigned base station (depot). Average travel speed and energy capacity might differ from vehicle to vehicle. The goal of the CMP is to calculate a sequence of routes for each vehicle that maximizes the number of visits to the points of interest, and minimizes the time delays between the visits during the given planning horizon.

The assignment of vehicles to their home stations in CMP steers the search process towards clustering the area. Even though this strategy can be very effective, this limits the solution space. Therefore, the requirement for vehicles to always return to their home station was relaxed in CMP with inter-depot routes (CMPID): vehicles can renew their energy capacity or finish their mission at any base station.

The CMPID with priorities (CMPIDP) includes different importance levels and does not impose an upper bound on the mission time like in the CMP and CMPID. The main differences between the three monitoring problems are summarized in Table 2.1.

	CMP	CMPID	CMPIDP
inter-depot routes	No	Yes	Yes
fixed mission time	Yes	Yes	No
priorities	No	No	Yes

Table 2.1: Comparison of the monitoring problems

There exist other important problems that also deal with periodic visitation. We discuss them in Section 2.5.

The following sections provide formal definitions of the three monitoring problems. The notation used for the definitions is listed in Table 2.2.

Table 2.2: Definition of the CMP and its extensions

Input	
G	graph that represents the area of interest
N	set of nodes
E	set of edges
N_b	set of base station nodes
N_p	set of picture point nodes
$d_{i,j}$	distance from node i to node j
lvt_p	time difference between the last visit at point p and the start of the mission
pr_p	priority of point p (CMPIDP)
$nBat_{b,t}$	number of batteries of type t at base station b
$nBat_t$	total number of batteries of type t
T	set of vehicle types
V	set of vehicles
$type_v$	type of vehicle v
$homeSt_v$	home station of vehicle v (CMP)
$inRemCap_v$	initial remaining energy capacity of vehicle v
$inLoc_v$	node where vehicle v is initially located
sp_t	average speed of vehicles of type t
$batCap_t$	maximal flight time defined by a battery capacity of vehicles of type t
$servT_t$	service time of vehicles of type t
$tChBat_t$	time to change a battery for vehicles of type t
mt	mission time (CMP, CMPID)
Solution	
R_v	sequence of routes of vehicle v
$R_{v,y}$	y-th route of vehicle v
nR_v	number of routes of vehicle v
$r_{v,y,\epsilon}$	node on position ϵ in route $R_{v,y}$
$t(R_{v,y})$	flight time of the route $R_{v,y}$
$art(\epsilon, R_{v,y})$	arrival time at node on position ϵ in route $R_{v,y}$
cmt_v	current mission time of vehicle v

Table 2.2: (Continuation) Definition of the CMP and its extensions

Goal function	
$f(x)$	goal function (CMP, CMPID)
A_p, B_p, C_p	summands of function $f(x)$
$artO_{z,p}$	arrival time at point p during its z-th visit
$nVis_p$	number of visits of point p
$f'(x)$	goal function (CMPIDP)
A'_p, B'_p, C'_p	summands of function $f'(x)$
mt_v	total mission time of vehicle v (CMPIDP)
$nChange_v$	number of battery changes of vehicle v (CMPIDP)

2.1 Continuous Monitoring Problem

Input: The *area* of interest is represented as a complete, weighted graph $G = (N, E)$, where N is the set of nodes and E is the set of edges connecting them. The set of nodes N_b represents the base stations, the remaining nodes N_p are the picture points. Every edge $e_{i,j} \in E$ has weight $d_{i,j}$ that represents the distance between the nodes i and j. In the presence of obstacles, this weight equals the length of the shortest path between the nodes around the obstacles. The triangle inequality must hold, i.e., $\forall i, j, l \in N, d_{i,j} + d_{j,l} \geq d_{i,l}$.

Every *picture point* $p \in N_p$ is characterized by the time difference lvt_p between the last visit and the start of the mission that indicates how long the point remained unvisited before the monitoring started. It equals zero for all points, if none of them have been visited before. If only some points have no previous observations, their lvt_p equals to the maximal lvt_p value of the visited points.

Every *base station* $b \in N_b$ is characterized by the number of batteries $nBat_{b,t}$ for a vehicle type $t \in T$. The total number of batteries of type t is denoted as $nBat_t$. The number of batteries can also be interpreted as the amount of any other kind of energy source, e.g., the amount of energy at a battery recharging facility or the amount of fuel.

Every *vehicle* $v \in V$ is characterized by its type $type_v$, home station $homeSt_v$ where it can renew its capacity, initial remaining battery capacity $inRemCap_v$ and initial location $inLoc_v$. The initial location can be either a picture point or a home station of the vehicle.

Vehicles of *type* $t \in T$ has an average travel speed sp_t, maximal flight time $batCap_t$ defined by a battery capacity, service time $servT_t$ (e.g., to take a picture or read the

sensory data), and time to renew the energy capacity $tChBat_t$ (e.g., changing the battery or refueling).

Finally, a mission time mt is given as the time by which all vehicles must complete their last flights.

Output: A solution to the described problem is a sequence of routes for each vehicle $v \in V$, i.e., $R_v = (R_{v,1}, ..., R_{v,nR_v})$, where nR_v is the number of routes of vehicle v. Every route is a sequence of nodes $R_{v,y} = (r_{v,y,1}, ..., r_{v,y,k})$, where k is the number of visits at the nodes made by vehicle v; $r_{v,y,2}, ..., r_{v,y,k-1}$ are picture points. Nodes $r_{v,y,1}$ and $r_{v,y,k}$ are the home station $homeSt_v$ except for the first point of the first route, which equals to the initial location of vehicle v, i.e., $r_{v,1,1} = inLoc_v$.

A solution is feasible if it fulfills the following constraints:

- the flight time of each route does not exceed the battery capacity;
- each node is visited by at most one vehicle at a time;
- a vehicle can change its battery only at its home station;
- at the time mt depicting end of the mission, all vehicles must be at their stations;
- a fleet cannot use more batteries than there are available.

The maximal flight time, limited by energy capacity, also limits the flight time for every route (capacity constraint). The flight time of y-th route of vehicle v of type t is a sum of the time needed to travel between its way points and the total service time:

$$t(R_{v,y}) = \left(\sum_{\epsilon=1}^{k-1} \frac{d_{r_{v,y,\epsilon}, r_{v,y,\epsilon+1}}}{sp_t} \right) + (k-2) \cdot st_t \ ,$$

where $d_{r_{v,y,\epsilon}, r_{v,y,\epsilon+1}}$ is the distance between consecutive points $r_{v,y,\epsilon}$ and $r_{v,y,\epsilon+1}$ of the route $R_{v,y}$, k is the number of points in the route, sp_t and st_t are the average speed and the service time of vehicles of type t, respectively.

Arrival time at ϵ-th point in route $R_{v,y}$ of vehicle of type t is a sum of the flight time of the preceding routes, the total time spent on battery changing, and the flight time within route $R_{v,y}$ until ϵ-th point:

$$art(\epsilon, R_{v,y}) = \sum_{y'=1}^{y-1} t(R_{v,y'}) + (y-1) \cdot tChBat_t + \left[\left(\sum_{\epsilon'=1}^{\epsilon-1} \frac{d_{r_{v,y,\epsilon'}, r_{v,y,\epsilon'+1}}}{sp_t} \right) + (\epsilon - 2) \cdot st_t \right] .$$

Two problem attributes are defined on the basis of arrival time. The first attribute is called current mission time cmt_v of vehicle v. It is used during route construction

and is defined as the time, when the vehicle would finish observing its last point in the current partial solution. Secondly, the goal function of the problem is expressed via arrival times at picture points.

Objective: The task is to provide an up-to-date information during the fixed period of time. The ideal solution would be to place a vehicle over each picture point in order to constantly observe all the points. However, due to the limited number of vehicles and large number of points, this solution is not possible in the real world. In this case, the problem requires a solution with maximal number of visits and minimal time lag between them.

The aforementioned goals are incorporated in the objective function that has to be minimized. It is based on the concept of penalties, i.e., negative rewards for every time delay. Given a feasible CMP solution x, the goal function is computed as follows:

$$f(x) = \sum_{p \in N_p} \left[A_p + B_p + C_p \right] \ . \tag{2.1}$$

The first summand A_p of the function penalizes the time delay until the first visit at point p. If the point was observed before the monitoring started ($lvt_p > 0$), then lvt_p is also considered in the penalty:

$$A_p = (lvt_p + artO_{1,p})^2,$$

where $artO_{1,p}$ is the first arrival time at picture point p.

The second summand B_p summarizes penalties for the time delays between all visits to the point:

$$B_p = \sum_{z=1}^{nVis_p - 1} (artO_{z+1,p} - artO_{z,p})^2,$$

where $artO_{1,p}, ..., artO_{nVis_p,p}$ are arrival times for all visits to point p sorted in ascending order, $nVis_p$ is the number of times point p was visited.

Finally, the last part C_p penalizes the time delay from the last observation at the point until the end of the mission:

$$C_p = (mt - artO_{nVis_p,p})^2.$$

Figure 2.1 illustrates the three summands of the goal function (2.1).

A solution example is shown in Figure 2.2. It contains three vehicles each assigned to its own station with two spare batteries per vehicle. The first image depicts the first route of each vehicle starting at its initial location. The remaining two images

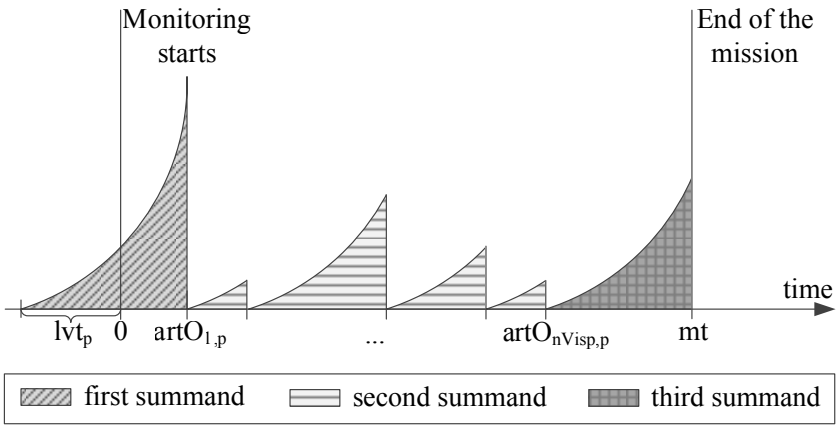
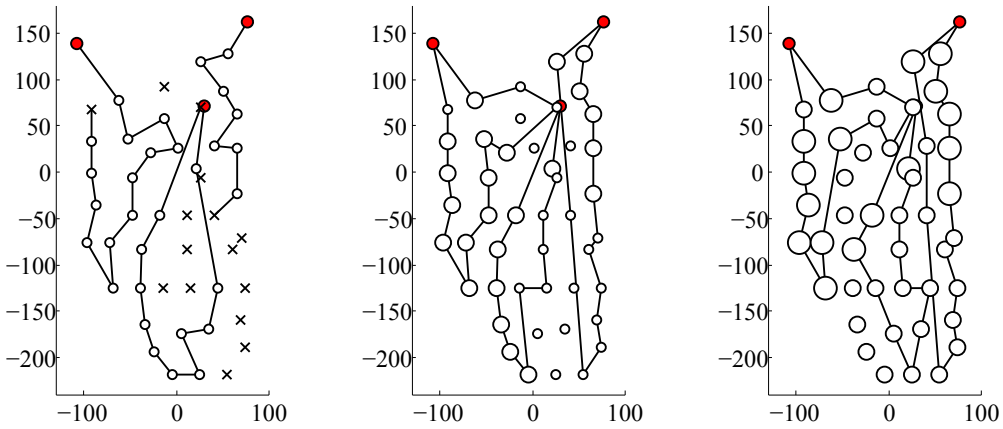


Figure 2.1: Visual interpretation of penalties of point p.

demonstrate the routes that followed. When points have not been visited yet, they are depicted with sign "x", and with circle otherwise. The size of such circle is proportional to the number of visits at a point.



(a) first routes (b) second routes (c) third routes

Figure 2.2: A solution of a CMP instance.

An important characteristic of an efficient CMP solution is to consistently survey all the pictures points. In the previous example, this is illustrated by the even number of visits made at each point.

2.2 CMP with Inter-Depot Routes

Solutions to the continuous monitoring problem tend to form areas of interest around the home stations of the vehicles. Such solutions are not efficient when batteries or vehicles are not distributed evenly over the base stations. To resolve this, the CMP was extended to allow vehicles to change batteries at any station [56]. This variation of the monitoring problem is called CMP with inter-depot routes. The name was inspired by the similar extension of the vehicle routing problem.

The difference between the CMPID and CMP is the absence of home stations and the corresponding restriction. The other constraints and the goal function remain as described in the previous section.

2.3 CMPID with Priorities

There are two essential differences between the CMP, CMPID and CMPID with priorities. The CMPID with priorities uses priority information and does not set an upper bound on the mission time. The first extension aims at providing information about the picture points at a frequency proportional to their priorities. This extension is of high importance: often, events of interest are not evenly distributed over the area. In this case, zones with more activity should be observed more frequently. The goal of the second extension is to utilize the available vehicles and energy resources so that as much information as possible is gained per time unit.

Input: The CMPIDP includes all parameters described in Section 2.1 except home stations $homeSt_v$ and mission time mt. Thus, initial location $inLoc_v$ of a vehicle v can be a picture point or any of the base stations, i.e., $inLoc_v \in N$. Additionally, every picture point p is assigned with priority pr_p that denotes its importance, i.e., the lower the priority value, the less important this point is.

Output: According to the mentioned changes, a feasible CMPIDP solution fulfills the following constraints:

- the flight time of each route does not exceed the battery capacity;
- each node is visited by no more than one vehicle at a time;
- a vehicle can change its battery only at the base stations with available spare batteries of the corresponding type;
- a fleet cannot use more batteries than there are available;

- every vehicle must finish its mission at a station.

Objective: The goal function of the CMP and CMPID penalizes time delays between visits starting from the beginning of the mission until the maximal possible end of the mission denoted as mt. As opposed to these two problems, the CMPIDP does not impose an upper bound on the mission time. Therefore, the penalties are computed until the end of the visiting schedule with additional penalties for the unused energy resources.

The end of the visiting schedule is estimated as the maximal possible mission time of a vehicle, given the number of batteries it used. The mission time of a vehicle v of type t is calculated as follows:

$$mt_v = nBatChange_v \cdot (batCap_t + tChBat_t) + inRemCap_v,$$

where $nBatChange_v$ is the number of times a vehicle has changed its battery, $batCap_t$ is the battery capacity of type t, $tChBat_t$ is the time to change a battery, $inRemCap_v$ is the initial remaining capacity of vehicle v.

The following example illustrates the computation of the mission time of a vehicle. A UAV can fly up to 5 minutes with its initial on-board capacity ($inRemCap_v = 5$). In the current solution it makes use of 3 additional fully-charged batteries with up to 15 minutes of flight time with each of them ($nBatChange_v = 3, batCap_t = 15$). Changing a battery takes 1 minute ($tChBat_t = 1$). Thus, the longest possible mission of this UAV will last $5 + 3 \cdot (15 + 1)$.

Taking the aforementioned computations into account, the goal function of the CMPIDP is as follows:

$$\begin{aligned} f'(x) &= \sum_{p \in N_p} \left[A'_p + B'_p + C'_p \right] \ , \\ A'_p &= \Big((lvt_p + artO_{1,p}) \cdot pr_p \Big)^2 \ , \\ B'_p &= \sum_{z=1}^{nVis_p - 1} \Big((artO_{z+1,p} - artO_{z,p}) \cdot pr_p \Big)^2 \ , \\ C'_p &= \Big((max_{v \in V}(mt_v) + batPenalty - artO_{nVis_p,p}) \cdot pr_p \Big)^2, \end{aligned} \tag{2.2}$$

where A', B', C' are summands similar to A, B, C in (2.1) proportional to priorities, pr_p is the priority of point p, ($max_{v \in V}(mt_v)$ is the estimated end time of the mission, mt_v is the mission time of vehicle v, $batPenalty$ is the sum of penalties for all unused batteries and equals their total capacity. If all batteries are utilized, $batPenalty = 0$.

This goal function is close to the function (2.1) but with a conceptual difference. The CMP and CMPID aim at distributing visits evenly over a fixed mission time. For the CMPIDP, the mission time is not given. Thus, by minimizing the time lag between the visits, it minimizes the overall mission time as well. The following example demonstrates this difference. Two vehicles have to monitor three target points with four batteries. Fig. 2.3 demonstrates the assignment of batteries to vehicles and the ideal visit distribution for both problems. Visits of the CMP/CMPID solution are allocated farther from each other in order to cover a fixed planning horizon. The solution of the CMPIDP, in contrast, achieves shorter delays by using both vehicles simultaneously.

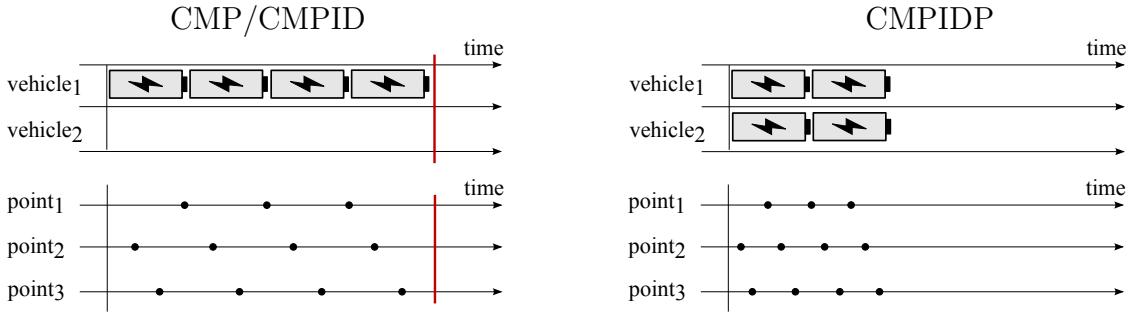


Figure 2.3: Solution parameters for the problems CMP, CMPID and CMPIDP. The top charts show assignments of the batteries to the vehicles. The bottom charts demonstrate the corresponding distribution of visits. Red lines indicate fixed, given mission time for the CMP and CMPID.

2.4 Complexity of the Problems

In this section, we derive the complexity of the CMPIDP. This property of the other two problems, CMP and CMPID, can be derived in the same way.

The CMPIDP is an NP-hard problem. This property can be derived by a Turing reduction to the traveling salesman problem (TSP) [34]. Let us consider a CMPIDP with one base station, one vehicle v of type t and one battery with capacity $batCap_t$. Suppose $S[G, V, T, C]$ is a subroutine for solving the CMPIDP, where G is a complete graph representing an area, V is a set of vehicles, T is a set of types, and C is the set of battery capacities with $batCap_t \in C$. Note, there is a minimal battery capacity $batCap_t$, which allows a UAV to visit every picture point exactly once and to return to the base station. Such a route corresponds to a TSP-route that minimizes the overall travel time as well as $max_{v \in V}(mt_v)$ and, as a consequence, the CMPIDP goal function. Given the minimal distance $minD$ between points, the following algorithm

constructs a battery capacity by a polynomial number of calls of S s.t. a TSP-route is generated.

1. Set $batCap_t^* \leftarrow minD$.
2. If $S[G, V, T, C]$ returns a route visiting all points, halt.
3. Set $batCap_t^* \leftarrow batCap_t^* + minD$ and go to Step 2.

This procedure uses a polynomial number of calls of S and, thus, would be a polynomial time algorithm for solving the TSP, if S were a polynomial time subroutine for CMPIDP. Consequently, the CMPIDP is NP-hard and cannot be solved in polynomial time unless P=NP.

2.5 Related Problems

There exist numerous routing problems with various constraints and objectives. In this section, we discuss the most related problems and outline their solving methods. Parameters of these problems are summarized in Table 2.3. The main state-of-the-art solving approaches are listed with the corresponding references in Table 2.4.

Significant amount of research is conducted on the route planning problems in the area of operational research (OR). One of the most well-known problems in this domain is the vehicle routing problem (VRP), where a fleet of vehicles has to visit a set of points and return to a station. Typically the total traveling time is optimized. There exist various extensions of this problem [89], [28]. The most related extensions are described below. They include problems with refueling possibilities, time-based features that can be used to model monitoring, and some works from the related application domains such as robotic patrolling and aerial surveillance. More information about the other VRP variants and their solving techniques can be found in [89] and [91].

Table 2.3: Parameters of related problems grouped according to the closest classical problem

Reference	Vehicles							Depots				TW		Points				Objective
	multiple	heterogeneous	fixed number of vehicles	limited capacity for goods	limited energy capacity	multiple trips	inifinite number of trips	multiple	inter-depot routes	fixed planning horizon	periodic	single per point	multiple	demands on goods	site-dependent	priorities	multiple visits	
Continuous monitoring problem (our work)																		
[54]	+	+	+	-	+	+	-	+	-	+	-	-	-	-	-	-	+	sum of squared time delays
[57]	+	+	+	-	+	+	-	+	+	+	-	-	-	-	-	-	+	
[58]	+	+	+	-	+	+	-	+	+	-	-	-	-	-	-	+	+	sum of weighted squared time delays
Periodic vehicle routing problem																		
[49]	-	-	+	-	+	+	+	+	+	+	+	-	-	-	-	-	+	cost of fuel
[90]	+	-	-	+	+	-	-	+	-	+	+	-	-	+	-	-	+	total travel time
[10]	-	-	+	+	+	+	-	-	-	+	+	-	-	+	-	-	+	
[15]	+	+	+	-	+	-	-	-	-	+	+	+	+	-	+	+	+	weighted sum of total travel cost, overtime and workload balance
[4]	+	+	+	+	+	+	-	-	-	+	+	-	-	+	+	-	+	total operation cost
Vehicle routing problem with time windows																		
[24] (MDVRP)	+	-	+	+	+	-	-	+	-	+	-	+	-	+	-	-	-	total travel time
[24] (PVRP)	+	-	+	+	+	-	-	+	-	+	+	-	+	+	-	-	+	
[27]	+	+	+	+	-	+	+	-	-	+	-	+	-	+	-	-	-	traveling and deployment costs
[31]	+	-	+	+	+	-	-	-	-	+	+	+	-	+	-	-	+	weighted sum of travel time, waiting time and cost of used vehicles
[12]	+	+	+	+	+	-	-	+	-	+	-	-	+	+	-	-	-	
[6], [7]	+	-	+	-	+	+	+	-	-	+	-	+	-	-	-	-	-	weighted difference between total travel distance and rewards for visited customers
[42]	+	-	+	+	+	+	+	-	-	+	-	+	-	+	-	-	-	total operation cost

Table 2.3: (Continuation) Parameters of related problems grouped according to the closest classical problem

Reference	Vehicles							Depots		fixed planning horizon	periodic	TW		Points				Objective
	multiple	heterogeneous	fixed number of vehicles	limited capacity for goods	limited energy capacity	multiple trips	inifinite number of trips	multiple	inter-depot routes			single per point	multiple	demands on goods	site-dependent	priorities	multiple visits	
Vehicle routing problem with intermediate facilities																		
[5]	+	-	+	+	+	+	+	+	+	+	+	-	-	+	-	-	+	total travel distance
[25]	+	-	+	+	-	+	-	+	+	+	-	-	-	+	-	-	-	mission time
[88]	+	-	+	+	+	+	+	+	+	+	-	-	-	+	-	-	-	total travel time
[65]	+	-	+	+	-	+	-	+	+	+	+	-	+	+	+	-	+	distance traveled per day
[46], [14]	+	-	+	+	+	+	+	+	+	+	-	+	-	+	-	-	-	number of vehicles and overlapping routes, travel time, workload deviation
[40] (PVRP-IF)	+	-	+	+	+	+	+	+	+	+	+	-	-	+	-	-	+	total travel cost
[40] (MDVRPI)	+	-	+	+	+	+	+	+	+	+	-	-	-	+	-	-	-	
[30]	+	-	+	-	+	+	+	+	+	-	-	-	-	-	-	-	-	total travel distance
[79]	+	-	+	+	+	+	+	+	+	-	-	+	-	+	-	-	-	number of vehicles, total travel distance
[32]	+	-	+	+	+	+	+	+	+	-	-	-	-	+	-	-	-	total recharge cost
Patrolling task																		
[52], [3], [77]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	average/maximal time delay
[29]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	min. average time delay, max. minimal delay, min. delay variance
[1]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	+	+	weighted average event detection
[81]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	maximal time delay

Table 2.3: (Continuation) Parameters of related problems grouped according to the closest classical problem

Reference	Vehicles							Depots		fixed planning horizon	periodic	TW		Points				Objective
	multiple	heterogeneous	fixed number of vehicles	limited capacity for goods	limited energy capacity	multiple trips	inifinite number of trips	multiple	inter-depot routes			single per point	multiple	demands on goods	site-dependent	priorities	multiple visits	
Patrolling task (Continuation)																		
[71]	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	maximal time delay, mean square time delay
[53], [20]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	+	+	max. weighted reward over time
[83]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	+	+	max. reward over time
[19]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	deviation from the prescribed visiting frequency
[2]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	+	+	maximal weighted time delay
UAV routing problem																		
[43]	+	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	+	average event detection
[62]	+	+	+	+	+	-	-	+	+	+	-	-	+	+	+	+	+	min. changes to the given plan, number of vehicles, total travel time, max. effectiveness
[80]	+	+	+	+	+	-	-	-	-	-	-	-	-	+	-	+	-	max. total weighted service
[66]	+	+	+	-	-	-	-	+	+	-	-	-	-	-	-	-	-	total travel cost
[69]	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	+	+	total time-averaged risk
[93]	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	+	min. maximal time delay, max. number of observed events
[84]	+	-	+	-	+	+	+	-	-	+	-	-	+	-	-	+	+	total travel cost
[64]	+	-	+	-	+	+	+	+	+	+	-	-	-	-	-	-	+	maximal idleness
[85]	-	-	-	-	+	+	+	+	+	-	-	-	-	-	-	-	-	total travel cost

Algorithms	References
Construction heuristics	
Adaptive memory principle	[25]
Approximation algorithms	[2]
Auction-based/negotiation mechanisms	[3], [71]
Clarke and Wright (savings) algorithm	[54], [30]
Clustering + TSP-based strategy	[46], [30], [79], [1], [81]
Dynamic programming	[43]
Heuristic based on spanning tree	[29]
Insertion heuristic	[54], [57], [27], [5], [65], [46], [14], [88], [79], [32], [3], [1], [80], [69], [64]
Lin-Kernighan-Helsgaun heuristic	[66], [85]
Multi-stage heuristics	[49], [15], [83]
Problem transformation	[66]
Reinforcement learning based on Markov decision processes	[3], [77], [53], [20]
TSP-based patrolling strategy	[3], [29], [43], [69], [93]
Improvement heuristics	
Adaptive large neighborhood search	[7]
Ant colony optimization	[31], [19]
Genetic algorithm	[90]
Guided local search	[65], [88]
Local search	[30], [32], [85]
Simulated annealing	[27], [46], [32], [69]
Tabu search	[24], [10], [15], [4], [12], [5], [25], [14], [88], [79], [80]
Threshold accepting	[65]
Variable neighborhood search	[54], [57] , [12], [65], [14], [40], [88], [79]
Exact approaches	
Branch-and-price	[6], [42],[84]
Mixed-integer (linear) programming	[62]

Table 2.4: Solving methods for the related problems (in alphabetical order)

Multi-trip vehicle routing problem (MTVRP) is also know as vehicle routing problem with multiple use of vehicles. It is a version of the VRP, where every vehicle is allowed to have multiple routes [87]. The maximal possible number of routes per vehicle is limited. Multi-trip extension can be used to model multiple routes of UAVs, when multiple spare batteries are given. However, in the MTVRP every customer is visited exactly once in contrast to the CMP and its extension, in which a customer

can be visited multiple times. Two important extensions of the MTVRP are the MTVRP with time windows and the periodic MTVRP. They can be used to model multiple visits and will be discussed further.

To the best of our knowledge, only a few publications address the pure multi-trip VRP. They propose heuristic methods such as tabu search [87], genetic algorithms [76], or adaptive memory programming [67].

Periodic vehicle routing problem (PVRP) is a variant of the VRP, where periodical visits at customer locations are required [22]. Given a set of time units (e.g., days or hours), each target point has a set of possible visit schedules (e.g., a point should be visited either on Monday and Thursday or on Tuesday and Wednesday). At the end of each time unit (e.g., working day) vehicles return to their depots. A solving algorithm should not only build routes for the given target points but also select one of the schedules for each customer and fulfill customers demands. Typically the total travel time is minimized. One can model monitoring problem as a periodic VRP with one full coverage of an area representing one time unit. However, this approach has a disadvantage. According to the definition of the PVRP, vehicles have to return to their stations at the end of each coverage/time unit. This policy is not efficient, since the fleet does not utilize all available energy capacity. Additionally, each vehicle performs at most one trip per time unit, i.e., refueling is not allowed. Visit frequency is given for each customer and cannot be changed or optimized.

State-of-the-art methods for the PVRP include methods such as exact algorithms [8], a combination of variable neighborhood search and simulated annealing [41], tabu search [24], genetic algorithm [90]. A good and relatively recent survey on the PVRP extensions and the state-of-the-art methods is provided in [33].

Vidal et al. [90] described a *PVRP with multiple depots*. Vehicles have limited travel time and capacity for carried goods. Each vehicle is assigned to a depot, where it starts and ends each route, at most one per time unit. All routes visiting the same customer must originate at the same depot. The goal is to construct routes fulfilling the constraints and minimizing the total travel time. Vidal et al. proposed a hybrid genetic algorithm that was evaluated on several problems: MDVRP, PVRP and MDPVRP.

The problem of *persistent visitation with fuel constraints* presented in [49] is a relaxation of the aforementioned problem that includes multiple trips and inter-depot routes. Inter-depot routes extension allows a vehicle to refuel its tank at any of the given facilities. The amount of fuel at facilities is not limited. Periodicity is defined by a given deadline for every visit of every point. The goal is to minimize the cost of

purchased fuel. Las Fargeas et al. [49] proposed a three-step approach for a single-vehicle problem: find all possible tours that satisfy the deadlines, find the optimal amount of purchased fuel for each tour and choose the solution with minimal cost.

There are several case studies based on the PVRP. The case study [10] of delivering linen to different clinics was modeled as a variation of the PVRP with a single vehicle and multiple trips per day. The authors implemented a tabu search algorithm with GENIUS heuristic for customer insertion and removal [22]. The algorithm allows infeasible solutions but penalizes every constraint violation. Blakeley et al. [15] describes a complex PVRP with multiple constraints imposed by a company, e.g., time windows for the customer service as well as working hours of the service men for each day, different types of service. The problem is solved by a multi-stage heuristic based on tabu search.

Site-dependent multi-trip periodic vehicle routing problem was presented by Alonso et al. [4]. Every customer is characterized by the possible visit schedules, demand of goods per service and a set of feasible vehicles that can provide required service. Every vehicle has a certain operation cost per distance unit, maximal allowed number of trips per day, limited capacity and travel time per route. The objective of the problem is to minimize the total operation cost. Alonso et al. proposed a modified tabu search similar to [10].

In **vehicle routing problem with time windows** (VRPTW), each customer has a time interval during which a vehicle can provide service [82]. The interval is defined by its lower and upper bound. If a vehicle arrives earlier than the lower bound, it has to wait until the service is allowed to start. Arrivals later than the upper bound are prohibited. Amount of goods that a vehicle can carry is limited. The optimization criteria can be the total waiting time, total travel time, number of used vehicles. The VRPTW can describe the CMP by introducing time windows for every visit of a point. The delays between the time windows should be identical and equal to the required update frequency. However, it is hard to estimate this frequency beforehand, while incorrect estimation leads to inefficient solutions or even to no feasible solution.

The VRPTW received a significant amount of attention, due to its practical importance. Application domains include, for example, ware delivery, waste collection and repair services. The common solving methods are tabu search [24], evolutionary algorithms [73], iterative local search [38], and variable neighborhood search [16]. Recently a promising exact method was proposed in [9]. More details on the state-of-the-art approaches and their comparisons are provided in [17], [18] and [36].

Cordeau et al. [24] described a *PVRP with time windows* as well as a *multi-depot VRPTW*. In the second problem, every vehicle must be assigned to a depot, where it starts and finishes every route. Both extensions of the VRPTW were solved by a tabu search algorithm. To switch between solutions, the algorithm simply removes a point from a route and inserts it into another route. The search is diversified by allowing infeasible solutions.

Despaux and Robledo [27] described a variant of the VRPTW: *multi-trip VRP with time windows and heterogeneous fleet*. Every vehicle has a certain capacity for goods it can carry, traveling cost for moving between the customers, and fixed cost when it is used. Multiple trips are permitted, which allows modeling problems with high customer demands and low trunk capacity. All routes must be finished by the end of the given period. The minimization objective function comprises of traveling and deployment costs. Despaux and Robledo suggested insertion heuristic to construct an initial solution and simulated annealing to improve it. Further we refer to a heuristic as "insertion heuristic" if it constructs a solution by adding one point at a time, and selects a point and/or position in a route based on some metric, for example, shortest distance.

Another related VRPTW extension is the *periodic VRP with multiple time windows and multiple visits* [31]. Instead of only one time window, each customer point has multiple time windows and a required number of visits as opposed to the CMP, where the number of visits has to be defined. In addition, the planning horizon is split into subperiods and every route must start and end within one subperiod as in the PVRP. A subperiod can be interpreted as a battery capacity, as Favaretto et al. [31] did not explicitly define this constraint. The goal is to minimize the sum of weighted waiting time, travel time and fixed cost of every used vehicle. The problem was solved by using ant colony optimization algorithm. Later Belhaiza et al. [12] proposed a hybrid heuristic based on the variable neighborhood search and tabu search. This heuristic outperformed the method of Favaretto et al. on almost all instances.

Hernandez et al. [42] described a *multi-trip VRPTW*, where the goal is to minimize the total operation cost. They proposed a two-step exact algorithm based on the branch-and-price approach. Azi et al. [6] used a different goal function. Due to strict constraints on time windows and maximal travel time, it might be infeasible to visit all customers at least once. Therefore, every customer is assigned a reward for being visited. The goal is to minimize the difference between the total travel distance and the rewards for visited customers. Azi et al. suggested an exact algorithm, branch-

and-price with column generation. Later Azi et al. [7] described a heuristic approach called adaptive large neighborhood search.

The next set of problems is grouped under the title **vehicle routing problem with intermediate facilities** (VRP-IF). The VRP-IF is a variant of the multi-depot vehicle routing problem where, in addition to depots, there are intermediate replenishment facilities to renew capacity of the vehicles. The goal is to minimize the total travel distance.

Angelelli and Speranza [5] studied a *periodic VRP-IF*. They solved the problem by constructing an initial solution with one of the three different insertion heuristics and improving this solution with a tabu search.

Crevier et al. [25] introduced a *multi-depot VRP with inter-depot routes* (MD-VRPI). It can be seen as a VRP-IF where depots play the role of intermediate facilities. Crevier et al. proposed a heuristic approach based on the adaptive memory principle for constructing an initial solution and a tabu search for improving it. Tarantilis et al. [88] suggested a hybrid metaheuristic based on the variable neighborhood search, tabu search and guided local search. An insertion heuristic is used as a construction heuristic.

A *waste collection problem* is a special case of the PVRP-IF. The objective is to collect the waste from the customers and deliver it to the intermediate facilities. Before arriving at the base station each vehicle has to visit an intermediate facility and empty its storage. Nuortio et al. [65] proposed an insertion heuristic and a guided variable neighborhood thresholding algorithm for a real-life scenario. Kim et al. [46] suggested an insertion heuristic and a cluster-based algorithm as construction heuristics and a simulated annealing as a metaheuristic. Later this algorithm was outperformed by the methods proposed by Benjamin and Beasley [14]. They construct the routes with an insertion heuristic and then improve them by either a tabu search or a variable neighborhood search, or their combination. Hemmelmayr et al. [40] proposes a hybrid method based on a combination of a variable neighborhood search and dynamic programming. It performs well on scenarios for both PVRP-IF and MDVRPI.

Green VRP (GVRP) is a variation of the VRP that aims at minimizing energy consumption and, as a result, its negative effect on the environment [50]. The state of the art proposes two ways to optimize energy consumption. The first way is to express fuel consumption via consumption rate per distance unit and optimize the total travel distance multiplied by the rate, e.g., [92], [48]. Such formulation is similar to the classical VRP and, thus, does not include such significant extensions as capacity

renewal and multiple visits. Another way is to introduce recharging possibilities and minimize the total purchased fuel [30]. This variation of the GVRP includes a depot, a set of target points and recharging facilities. When refueling, a vehicle can purchase any amount of fuel limited only by the size of its tank. The objective of the problem is to minimize the total distance traveled without violating fuel capacity constraint. Erdoğan and Miller-Hooks [30] solved this GVRP by using a local search heuristic with inter- and intra-route vertex exchange. It constructs initial solutions by using a modified savings algorithm or a combination of density-based clustering and savings algorithm. Schneider et al. [79] introduced an *electric VRP with time windows* that minimizes the number of used vehicles and total distance traveled. They solved the electric VRPTW by an insertion heuristic based on the spherical clustering and a combination of a variable neighborhood search and a tabu search. Felipe et al. [32] extended the GVRP by introducing different recharging technologies and a possibility of partial recharging. The extended problem minimizes the costs of purchased fuel with taking into account different prices during day and night. Initial solutions are constructed by using an insertion heuristic similar to the nearest neighbor heuristic with a random component. This solution is then improved with either a local search or a simulated annealing.

Apart from OR, similar problems were also considered in the field of robotics. In 2003 Machado et al. [52] introduced the **patrolling task** that is also referred to as repetitive sweeping/coverage, continuous area sweeping task, persistent monitoring/surveillance and multi-agent information gathering problem. The goal of this problem is to compute a set of routes to repetitively visit given points. There are several possible objectives, for instance, to minimize the average delay between visits or to maximize the minimal delay. There might be several extensions like priorities and changing edges between the points and their weights (dynamic obstacles). The planning horizon is typically infinite but also can be fixed. In most papers (e.g., [3], [52], [1], [83], [93], [19]) the capacity constraint is neglected, as typical missions do not outlast the energy capacity of the employed robots. The earliest and most popular approaches for the patrolling task are based on solving the traveling salesman problem (TSP), a single-vehicle VRP. One option is to build a route through all points and place the robots along it [3], [29]. The other approach is to split the area into clusters, one for each vehicle, and solve the TSP for each cluster [1], [81]. For clustering, Smith and Rus [81] used k-medians algorithm, whereas Ahmadi et al. [1] suggested a negotiation-based algorithm. Recently auction-based algorithms started to gain more attention [3], [71]. Some authors also applied reinforcement learning

and Markov decision process to modeling the system [77], [53], [20]. Stranders et al. [83] suggested a divide-and-conquer algorithm that also incorporates Markov decision processes. Additionally, authors described an algorithm for repairing the routes in case of vehicle's failure or graph changes. Finally, Cannata and Sgorbissa proposed a real-time ant-like algorithm PatrolGRAPH [19].

A *min-max latency walk problem* described in [2] is a variation of the patrolling task. The points have weights depicting their priorities. The goal is to obtain a path that visits the points multiple times while minimizing maximal weighted time delay of a vertex. Alamdari et al. [2] proposed two approximation algorithms and as a test example simulated patrolling of the city of San Francisco.

There are two interesting works based on real robots. Iocchi et al. [44] evaluated some of the aforementioned state-of-the-art strategies on real testbeds. Pippin et al. [70] described the approach to monitor performance of the real robots during the mission, when patrolling routes are given. The obtained observations are used to reassign the tasks of the poorer performing robots to the others.

Early research on the **UAV routing problem** focused on minimizing the number of drones to survey an area once [75], [68], [45]. In our work we do not aim at minimizing the number of drones, unless it will improve the performance of the overall fleet. The mentioned papers consider wind and various mission-related constraints like no-fly zones (obstacles), service time, threat level, time windows, priorities, and heterogeneous fleet. Such problems were solved with Monte Carlo simulations [75], a reactive tabu search [68] or a combination of an insertion heuristic and several improvement heuristics based on a local search or a tabu search [45]. The last work incorporated the most of the mentioned constraints. Richards et al. [74] considered a similar problem that minimizes the mission completion time. They proposed two methods: a mixed-integer linear programming model and an approximate algorithm that makes an estimation of the flight time. The first approach is guaranteed to find the optimal solution but is applicable to only small-size scenarios. Evaluation instances used for this approach consist of only four points.

Murray and Karwan [62] described a *UAV routing problem with time windows, heterogeneous fleet and limited fuel capacity*. The goal is to recompute the given routes after an event so that the number of route changes, total travel time and number of vehicles are minimized, while the mission effectiveness is maximized. The problem was modeled with mixed integer programming and solved by CPLEX solver. The test instances contained only up to 13 target points.

A problem of routing Unmanned Combat Vehicles (UCVs) was considered in [80]. The problem has only a single station and a heterogeneous fleet with certain payload and fuel capacities. Each target has a priority as well as a minimal and maximal required service. The latter estimates the amount of ammunition required to destroy an enemy location. The goal is to maximize the total weighted service provided to the targets. The authors solved the problem by using an insertion heuristic and a tabu search.

Oberlin et al. [66] described a *heterogeneous, multiple depot, multiple UAV routing problem*, where a fleet of drones visits every target exactly once. An algorithm proposed by Oberlin et al. transforms this problem into the asymmetric TSP, a TSP with different edge weights depending on the travel direction ($\exists i, j \in N \;\; d_{i,j} \neq \;\; d_{j,i}$). Then the asymmetric TSP is solved with the Lin-Kernighan-Helsgaun heuristic. Finally, the obtained route is transformed into a solution for the original problem.

A few papers address a UAV or UCV monitoring problem and model it as the patrolling task. Park et al. [69] studied a single-UCV monitoring problem with priorities that minimizes the possibility of an enemy's infiltration within the fixed time horizon. The goal function is the same as in the CMP and CMPID. This problem was solved by building routes using one of seven different construction heuristics, insertion heuristics or TSP-based cyclic strategies, and iteratively improving them with a simulated annealing. Huynh et al. [43] described a problem with a similar objective but a different goal function. It minimizes the average event detection time. This problem was solved by using one of following policies: a biased tile sweep, a TSP sampling and a TSP sampling with receding horizon, or approximate dynamic programming. Another version of the UAV monitoring with one vehicle was proposed by Yu et al. [93]. Here every target point generates events at random and has some statistics of the past events. In addition to a route, the amount of time that a vehicle waits at each point has to be defined so that (1) the number of observed events is maximized and (2) the delays between sequential visits at every point are minimized. Yu et al. suggested a two-step approach: first a TSP route is computed, then an optimal schedule of waiting times is found using a gradient descent method.

A UAV routing problem in [84] was modeled as a *VRP with multiple time windows and multiple visits*. The UAVs can recharge at any station any number of times. The goal function minimizes the total travel cost. This problem was solved by an exact method, an adapted Lagrangian branch-cut-and-price approach.

A problem very close to the CMP, CMPID and CMPIDP was considered in [64]. The area is represented as a grid, where every cell has an initial time delay from

the previous visit. The UAVs have an endurance constraint and can recharge at any station. Service of a point requires no time, since the authors consider nonholonomic vehicles that cannot hover at one point. The goal of the problem is to minimize the maximal time delay of the cells over an infinite period of time. The main difference to the CMP and its extensions is that the vehicles can recharge an infinite number of times as opposed to the limited number of batteries in the CMP. The authors suggested a control policy that takes into account the endurance constraint and dynamics of the vehicles. Some evaluation tests were performed on a real testbed.

A UAV routing problem described by Sundar and Rathinam [85] was modeled as a single-vehicle VRP with a refueling constraint. A UAV must visit each target at least once without exceeding its capacity. The capacity can be renewed in multiple ground stations. The total travel cost is minimized. The problem was solved in two steps. First, an initial solution is found by constructing a TSP-route with Lin-Kernighan-Helsgaun algorithm and inserting charging stations whenever required. The second step is to improve the obtained solution by 2-opt, 3-opt or adjacent one point exchange.

2.6 Summary

State of the art in artificial intelligence and operational research communities considers a number of related problems. Most problems do not consider renewal of the energy capacity or allow an infinite number of renewals. Due to a number of reasons, in many application cases such as disaster area monitoring, infinite recharging of vehicles might not be feasible. The other essential issue is the goal function, as most of the problems apart from the patrolling task minimize a metric not related to maximal information gain, for example, total travel distance or number of vehicles. To the best of our knowledge, there exists no problem that covers all aspects of the aerial monitoring with limited energy resources at replenishment facilities. This thesis aims to fill this gap as an attempt to draw more attention to this real-life problem.

Chapter 3

Solution Procedures

As we mentioned earlier, the CMP and its extensions are generalizations of the vehicle routing problem, which is an NP-hard problem. Therefore, they are also NP-hard, and there exists no polynomial time algorithm to find the optimal solution. This implies that the real-life scenarios can be solved in reasonable time only by a heuristic approach.

The remainder of this chapter is organized as follows. We provide some general information on the related approaches in Section 3.1. We proceed with the solving methods for the CMP in Section 3.2, the CMPID in Section 3.3 and the CMPIDP in Section 3.4. Section 3.5 explains the constraint programming model, used to obtain the optimal solutions. We conclude with the details on applying the proposed heuristics to the problems with dynamic environment in Section 3.6.

3.1 Related Heuristics

3.1.1 Clarke and Wright Algorithm

This heuristic (that is also known as savings algorithm) was introduced by Clarke and Wright in 1964 [21] and is still widely used due to its simplicity and short computational time. It solves the vehicle routing problem with heterogeneous fleet, where trucks deliver the requested merchandise to a large number of customers from a single depot. The goal is to construct such routes (at most one per vehicle) that fulfill demands of the customers with minimal traveled distance and do not exceed capacities of the vehicles.

The method is based on a 'savings' value that stands for the distance-wise benefit from connecting two customers in one route. The algorithm starts with single-customer routes starting and finishing at the depot. Then the savings value is com-

puted for every pair of customers i and j as the distance saved when the customers are visited in one route:

$$s_{i,j} = (d_{0,i} + d_{i,0} + d_{0,j} + d_{j,0}) - (d_{0,i} + d_{i,j} + d_{j,0}) = d_{i,0} + d_{0,j} - d_{i,j} \ ,$$

where $d_{0,i}, ..., d_{j,0}$ are the distances between two points, point 0 is the depot, points i and j are the customers. The routes before and after connecting customers i and j as well as their travel distances are depicted in Figure 3.1. Non-directed edges depict single-customer routes.

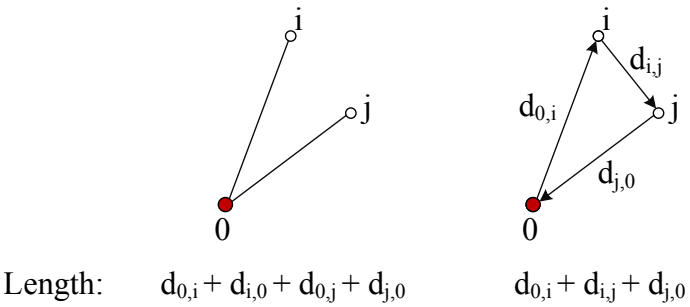


Figure 3.1: Routes before and after the points i and j are connected.

The remaining steps of the savings algorithm are as follows. All possible pairs of customers are ordered by their savings value in descending order. Then the algorithm iteratively joins the routes. For this, it takes the next not explored pair of customers with the maximal savings value. It connects these points if:

- they belong to different routes;
- both points are connected to a depot via at least one direct edge;
- after removing the two routes, there is a vehicle with sufficiently large capacity for the new joint route.

Figure 3.2 illustrates the workflow of this algorithm on a small example with six points and one vehicle. Savings values are reported in the savings table and colored so that the higher values have darker shade of gray. First, the customers 4 and 5 are connected, as they have the highest savings value. The next highest savings value belong to customers 2 and 4 that are therefore connected next. The third highest savings value has a pair $3 - 4$. However, point 4 violates the seconds condition, as it is in the middle of the route. Thus, these two points are not connected and the algorithm proceeds to the next pair. The last image depicts the complete solution.

A table with savings values

Nodes	*1*	*2*	*3*	*4*	*5*	*6*
1	0.00	18.76	7.20	6.79	1.83	0.56
2	18.76	0.00	26.41	34.54	18.60	2.58
3	7.20	26.41	0.00	31.93	23.76	7.37
4	6.79	34.54	31.93	0.00	36.46	11.62
5	1.83	18.60	23.76	36.46	0.00	19.36
6	0.56	2.58	7.37	11.62	19.36	0.00

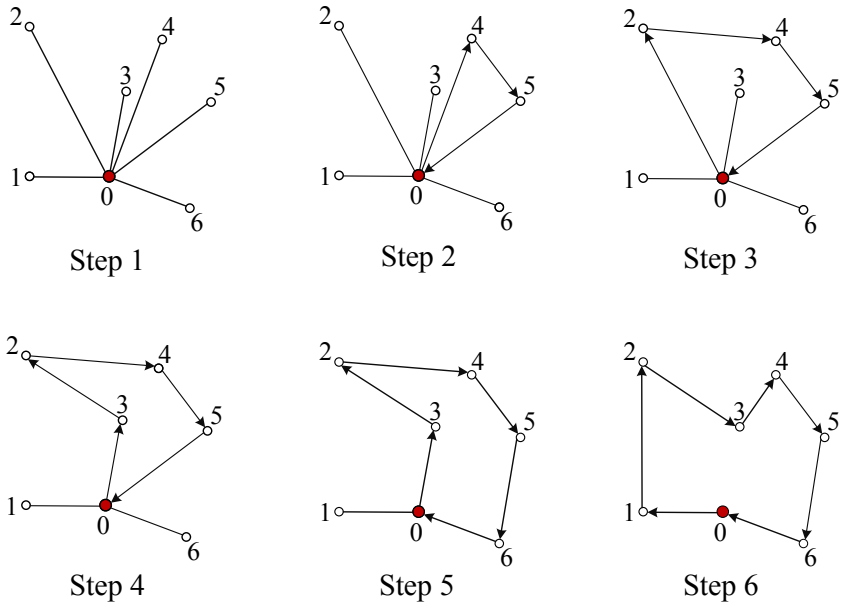


Figure 3.2: Sequential route construction by the Clarke and Wright algorithm.

3.1.2 Solomon's Insertion Heuristic

Solomon proposed a heuristic approach for a vehicle routing problem with time windows in 1987 [82]. In his formulation, the problem minimizes a weighted sum of the total travel distance and waiting time. Vehicles have limited capacity but the number of vehicles is unlimited.

As described in [82],the insertion heuristic constructs one route at a time by inserting one point after the other until all points have been visited. The following steps are used:

1. Start a new route with a selected customer. Solomon suggested several selection parameters such as a maximal travel distance or an earliest deadline;

2. For every unrouted customer chose the best feasible insertion position that does not violate the capacity and time windows constraints. The most efficient selection criteria of Solomon combines distance- and time-based parameters.

Given a customer u and a position between consecutive customers i and j, the distance-based parameter is computed as follows:

$$c_{11}(i,u,j) = d_{i,u} + d_{u,j} - \mu \cdot d_{i,j} \ , \ \mu \geq 0 \ ,$$

where $d_{i,u}$, $d_{u,j}$ and $d_{i,j}$ are the distances between the corresponding customers, μ is a coefficient of the algorithm. The parameter $c_{11}(i,u,j)$ shows how much the travel distance of the route would increase if the point u is inserted.

The time-based parameter defines how much later the service at the customer j will start after the insertion:

$$c_{12}(i,u,j) = b_j^{new} - b_j \ ,$$

where b_j and b_j^{new} are the service start times of the customer j before and after the insertion.

Finally, the best position for the customer u is between consecutive customers $i*$ and $j*$ that minimizes the following function:

$$c_1(i,u,j) = \alpha_1 \cdot c_{11}(i,u,j) + \alpha_2 \cdot c_{12}(i,u,j) \ , \ \alpha_1 + \alpha_2 = 1 \ , \ \alpha_1 \geq 0, \alpha_2 \geq 0,$$

where α_1 and α_2 are the coefficients defining the importance of each parameter.

3. Select the best feasible unrouted customer. For instance, the following metric can be used:

$$c_2(i,u,j) = \lambda \cdot d_{0,u} - c_1(i,u,j) \ , \ \lambda \geq 0,$$

where $d_{0,u}$ is the distance from the depot to the candidate customer u, λ is a coefficient of the algorithm.

It interesting that if $\alpha_1 = 1, \alpha_2 = 0, \mu = 1, \lambda = 1$ then this function equals to savings from inserting u between customers i and j.

4. If a customer was selected, insert it at the corresponding best position and go to step 2. If all points are scheduled, return the obtained solution; otherwise, go to step 1.

The criteria c_1 and c_2 are also called evaluation functions. These functions should reflect the optimization criteria of the problem but are not equal to it. The reason for that is that they measure quality of a partial solution that is required for route construction. The goal function of a problem, on the contrary, evaluates the complete solution and therefore cannot be used for constructing the routes.

3.1.3 Variable Neighborhood Search

The first works on the variable neighborhood search (VNS) were presented in [59] and [60]. It is a metaheuristic that includes multiple neighborhood operators instead of a single one to systematically explore solution neighborhoods. During the exploration a local search routine refines solutions obtained by the neighborhood operators.

The VNS starts with an initial solution obtained by using a construction heuristic and iteratively improves it. A set of operators N_k ($k = 1, \ldots, k_{max}$) can be applied to the current state (current solution) resulting in a set of successor states. The set of successor states generated by applying an operator N_i is a neighborhood of the current solution. A state is randomly selected from the neighborhood and improved by local search techniques. Then an acceptance phase evaluates whether the current solution is accepted or not based on the achieved improvements. If a solution is not accepted, the neighborhood is switched to the next one, i.e., another operator is applied. In short, the basic VNS steps are as follows:

1. Obtain a feasible *initial solution* x_i by exploiting a construction heuristic. Select a set of operators with cardinality k_{max} which will be applied to modify a solution. Choose a stopping condition, e.g., a limit on computational time or on the number of iterations.

2. Initialize current solution $x \leftarrow x_i$. Repeat the following steps until the stopping condition is met:

   (a) Set index of operator $k \leftarrow 1$;

   (b) Repeat the steps below until $k = k_{max}$:

      i. *Shaking.* Generate a new solution x' by applying k^{th} operator to the current solution;

      ii. *Local search.* Apply a local search method to x' which outputs x'' as a local optimum;

      iii. *Acceptance phase.* If the local optimum x'' is better than the solution x or another acceptance criterion is met, set $x \leftarrow x''$ and continue to search starting with the first operator ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$.

This method and its variations were used for a large number of problems like the TSP and the VRP in the last two decades. For more details on the VNS-related state of the art reader is forwarded to a survey by Hansen et al. [37].

3.2 Solving Continuous Monitoring Problem

Let us describe the heuristics developed for the CMP. We use an approach, where the initial solution can be constructed by a modified Clarke and Wright algorithm (CW) or a queue-based insertion heuristic (QI). Then it is iteratively improved by using a variable neighborhood search (VNS).

3.2.1 Modified Clarke and Wright Algorithm

The original Clarke and Wright algorithm (see Section 3.1.1) was proposed for the vehicle routing problem with restricted vehicle capacity and unlimited number of vehicles. There are several aspects in which the CMP differs from this problem (see Table 3.1).

Vehicle routing problem	Continuous monitoring problem
• single depot • every point is visited exactly once • unlimited number of vehicles • every vehicle has at most one route	• multiple depots • multiple visits at every point • limited number of vehicles • every vehicle can have multiple trips, number of trips is limited by the number of available batteries

Table 3.1: Differences between the CMP and the vehicle routing problem described by Clarke and Wright

The following extensions are incorporated into the savings algorithm to deal with the differences listed above:

- *for multiple depots*: savings value for every pair of points (i, j) is calculated with respect to the closest base station $x \in N_b$, i.e., $x \leftarrow arg\ \ min_{x' \in N_b}(d_{x,i} + d_{x,j})$;
- *for multiple visits of points*: routes for every coverage of the whole area are computed separately by applying the savings algorithm;
- *for multiple trips, limited number of batteries and vehicles*: the savings algorithm is applied until no more batteries are available or mission time is reached for all vehicles. Initially the modified savings algorithm starts with an empty set of routes as opposed to single-point routes like in the original savings algorithm. Later, when two points i and j are included in route $station - i - j - station$, the route is assigned a vehicle that can follow this route without exceeding its capacity. If there is no such vehicle, this route is not created.

Algorithm 1 describes the main steps of the modified Clarke and Wright algorithm: compute and sort savings values S in descending order (lines 2–8), build routes for one coverage at a time (lines 9–28). As explained above, savings values are calculated with respect to the closest base station.

Algorithm 1: Modified Clarke and Wright algorithm

input : problem description
output: a set of routes $R = \{R_v | v \in V\}$

```
1  R ← {} ;
2  S ← () ;
3  for i, j ∈ N_p (i ≠ j) do
4      x ← arg min_{x'∈N_b}(d_{x,i} + d_{x,j}) ;
5      s_{i,j} ← d_{i,x} + d_{x,j} − d_{i,j} ;
6      S ← S ∪ s_{i,j} ;
7  end
8  sortDecending(S) ;
9  repeat
10     R_cur ← {} ;
11     for s_{i,j} ∈ S do
12         r_i ← r_x ∈ R_cur, where i ∈ r_x ;
13         r_j ← r_y ∈ R_cur, where j ∈ r_y ;
14         if r_i = null and r_j = null then
15             v ← chooseVehicle(i, j) ;
16             if v ≠ null then
17                 r_new ← (homeSt_v, i, j, homeSt_v) ;
18                 R_cur ← R_cur ∪ r_new ;
19                 nBat_{homeSt_v, type_v}-- ;
20             end
21         end
22         else if violateNoConstraint(i, j, r_i, r_j) then
23             r_new ← join(i, j, r_i, r_j) ;
24             R_cur ← R_cur ∪ r_new ;
25         end
26     end
27     R ← R ∪ R_cur ;
28 until Σ_{t∈T} nBat_t = 0;
```

To construct routes for the current coverage R_{cur}, the algorithm sequentially considers every pair of points in the savings list. Depending on the status of the points, there are different conditions for connecting them:

1. *Both points of a savings pair are not in the routes R_{cur}.* They form a new route, if there is a vehicle that can visit them without violating any of the constraints. Such vehicle is selected by function CHOOSEVEHICLE(i, j) so that its current mission time is minimal, its capacity or maximal mission time are not exceeded and there is a spare battery to perform the new route.

2. *At least one savings point is in the routes R_{cur}.* The points are connected if they are not in the same route or in the middle of a route, and the joint route does not violate the capacity and maximal mission time constraints. This is ensured by function VIOLATENOCONSTRAINT(i, j, r_i, r_j) that returns *true* in case of success.

   The routes are merged by function JOIN(i, j, r_i, r_j). If both points are in the existing routes, these routes are merged. Then a vehicle requires only one spare battery instead of two and the needless battery is returned to the list of available batteries. If one of the points is not in R_{cur}, it is simply added to the route of the second point.

If neither of the conditions hold, no connection between the two points is added.

3.2.2 Queue-based Insertion Heuristic

Solomon introduced the original insertion heuristic for the vehicle routing problem with time windows (VRPTW) [82]. The CMP differs from the VRPTW in several aspects such as multiple visits per point and no time windows. Therefore, a new insertion heuristic has to be adapted to the CMP. In particular, it has to define a more suitable evaluation function and an order in which picture points will be inserted.

A good solution for the continuous monitoring problem is the one where the points are visited with nearly equal frequencies. Consequently, an efficient algorithm should avoid a situation where some points have significantly more visits than the others. Therefore, all picture points are placed in a queue $Q = (q_1, ..., q_m)$ that defines an order in which they will be inserted. In every step the first point of the queue Q is considered for insertion. Initially the points in the queue are ordered in ascending order by their value lvt_p, which is the time difference since the last visit.

Our evaluation function chooses a vehicle (1) whose last visited point is located as close as possible to the insertion point q_1 and (2) that can arrive at the insertion point the earliest. These parameters aim at optimizing the distance traveled by every

vehicle, and minimizing the time between visits at the insertion point, respectively. The evaluation function combines these conflicting goals by using a weighted sum:

$$g(q_1, v) = \alpha_1 \cdot d(v, q_1) \cdot scale + \alpha_2 \cdot art(v, q_1) \ , \ \ \alpha_1 + \alpha_2 = 1, \alpha_1 \geq 0, \alpha_2 \geq 0 \ ,$$

where q_1 and v are the point and vehicle considered for insertion, $d(v, q_1)$ is the distance between the last point visited by v and point q_1, $art(v, q_1)$ is the possible arrival time of vehicle v at point q_1. The *scale* coefficient brings the distance parameter to the same order of magnitude as arrival time and equals 10 to the corresponding power. The weights α_1 and α_2 reflect how much every parameter, i.e., distance and arrival time, influences the final decision.

Values of the coefficients should be chosen for a type of instances, i.e., number of vehicles, size of an area and presence of large obstacles. As soon as the values are selected for a type, no further tuning is required for new instances of the same type. The best values of the coefficients for some types are reported in Section 4.1.2.

Steps of the queue-based insertion heuristic are shown in Algorithm 2. First, the algorithm constructs a queue Q and initializes the first routes of the vehicles by inserting their initial locations (lines 1–3). After that, the heuristic iteratively extends the routes. In every iteration, if possible, the first point of the queue is added to the last route of the selected vehicle *bestVehicle*; otherwise, it is removed from the queue. The insertion loop terminates when there are no more points in the queue. Then the algorithm adds the final stations where UAVs will land and returns the obtained solution R.

The main loop (lines 4–30) starts by setting initial values to the following variables: vehicle $bestVehicle$ that will visit the first queue point q_1, the corresponding evaluation value $minValue$ and the flag *change* indicating whether a battery change is required.

The for-loop selects $bestVehicle$ among the vehicles that can visit the point q_1 without exceeding the mission time limit. Every vehicle is evaluated in several steps. First, if after visiting point q_1 a vehicle cannot reach its home station with its onboard capacity, it must replace its battery at the home station before visiting the point. This condition is indicated by the function NEEDBATCHANGE(v,R_v,q_1) returning *true*. Then, if 1) no other vehicle visits the point at the same time, i.e., NOCOLLISION$(v, R_v, q_1, homeSt_v) = true$, where the last argument is the station for a vehicle to change the battery before visiting point q_1 if required, and 2) the new evaluation value is better than the best so far, this vehicle is selected. The variable *change* is set to the value returned by the function NEEDBATCHANGE(v,R_v,q_1).

Algorithm 2: Queue-based insertion heuristic

input : problem description
output: a set of routes $R = \{R_v | v \in V\}$

$Q \leftarrow (q_1, ..., q_m)$ **s.t.** $\forall i \in [1, m-1]\,,\ lvt_{q_i} \leq lvt_{q_{i+1}},\ q_i, q_{i+1} \in N_p$;
$R \leftarrow \{\}$;
for $v \in V$ **do** $R_{v,1}$.ADD($inLoc_v$) ;
repeat
    $bestVehicle \leftarrow null$;
    $change \leftarrow false$;
    $minValue \leftarrow \infty$;
    for $v \in V$ **s.t.** NOMTIMEVIOLATION(R_v, q_1) **do**
        if NEEDBATCHANGE(v,R_v,q_1) **then**
            if NOCOLLISION($v, R_v, q_1, homeSt_v$) **and** $g(q_1, v) < minValue$ **then**
                $bestVehicle \leftarrow v$;
                $change \leftarrow true$;
                $minValue \leftarrow g(q_1, v)$;
            end
        else
            if NOCOLLISION($v, R_v, q_1, null$) **and** $g(q_1, v) < minEvalValue$ **then**
                $bestVehicle \leftarrow v$;
                $change \leftarrow false$;
                $minEvalValue \leftarrow g(q_1, v)$;
            end
        end
    end
    if $bestVehicle = null$ **then**
        $Q \leftarrow Q \backslash q_1$;
    else
        $Q \leftarrow (q_2, ..., q_{|Q|}, q_1)$;
    end
    if *change* **then** $R_{bestVehicle}$.INITIALIZENEWROUTE() ;
    $R_{bestVehicle,last}$.ADD(q_1);
until $Q = \emptyset$;
ADDFINALSTATIONS();
return R;

If no vehicle remain that can visit point q_1, then this point is removed from the queue; otherwise, it becomes the last point in the queue. This way algorithm ensures that the other points will be considered in the next steps.

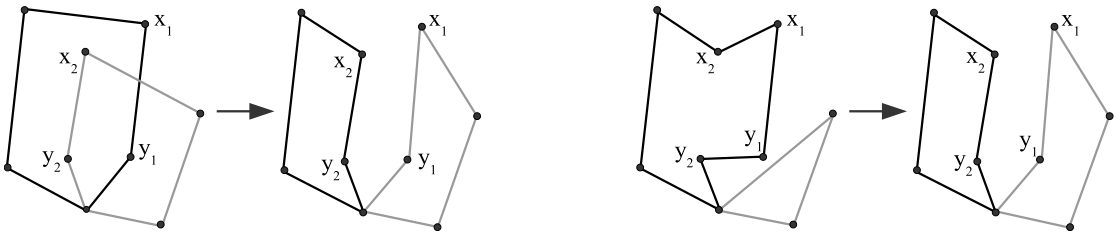
If insertion is feasible, the point is added to the last route of the selected vehicle. If required, a new route is initialized before the insertion so that the on-board capacity is renewed.

3.2.3 Variable Neighborhood Search

In order to apply the VNS to a specific problem or to a class of problems, the following three components should be defined: computation of an initial solution, modification procedures for shaking and local search steps, and acceptance strategy. An initial solution is computed by using either the CW or the QI heuristics described in the previous sections. The other two components are described below.

The purpose of the **shaking step** (see Section 3.1.3, step 2(b)i) is to modify the current solution in order to give way for improvements and escape from the local optimum. For this, various neighborhood operators are applied to the solution.

Cross-exchange [86] and *move* [78] operators are the most widely used inter-route operators [41]. The *cross-exchange* operator exchanges two equally long sequences of points of two routes. For instance, in Figure 3.3a, sequences $(x_1, ..., y_1)$ and $(x_2, ..., y_2)$ are removed from their initial routes r_1 (black) and r_2 (gray) and are inserted in the other routes r_2 and r_1, respectively. The *move* operator relocates a sequence of points from one route to another. Sequence $(x_1, ..., y_1)$ is removed from its initial route and is added to another route as shown in Figure 3.3b. Both operators choose routes, sequences and their lengths randomly. The sequence length can take any values up to the maximal allowed sequence length shown in Table 3.2 for every applied operator. Variable n is the shortest length of the two chosen routes; k is the sequential number of an operator (see Section 3.1.3).



a) *cross-exchange* b) *move*

Figure 3.3: Neighborhood operators for the shacking step

Monitoring is more efficient if picture points are visited more often. Therefore, in addition to the *cross-exchange* and *move* operators we introduce an *insert* operator which inserts points at random positions into a randomly chosen route. Table 3.2 shows the maximal possible number of insertion points for this operator. We use the order of the neighborhood operators that is determined by our experiments described in Section 4.1.2.

k	Operator	Maximal sequence length
1	*move*	min(1, n)
2	*move*	min(2, n)
3	*move*	min(3, n)
4	*cross*	min(1, n)
5	*cross*	min(2, n)
6	*cross*	min(3, n)
7	*cross*	min(4, n)
8	*cross*	min(5, n)
9	*cross*	min(6, n)
		Maximal number of points
10	*insert*	min(1, n)
11	*insert*	min(2, n)
12	*insert*	min(3, n)

Table 3.2: Neighborhood structure for the continuous monitoring problem, where k is the sequential number of an operator in the neighborhood

After the shaking step, every modified route is optimized by a local search (see Section 3.1.3, step 2(b)ii). For the **local search phase** we apply the commonly used 2-opt and 3-opt strategies depicted in Figure 3.4. These strategies are special cases of the k-opt [51], [39] algorithm.

The k-opt reconnects every combination of k arcs in a route in all possible ways so that the result is still a route. If the new route has a better cost value, it is accepted. In our implementation, travel distance of a route is used as a cost function, as it reduces the energy consumption and, consequently, more points can be inserted in the route. Since this cost function is based only on the local information about a single route, it is very time-efficient. The reconnection procedure is applied until there are no more possible improvements. The obtained solution is called k-optimal.

The computational time of the k-opt increases exponentially with k. However, larger k rarely are beneficial. Therefore, it is a common practice to apply only 2-opt and 3-opt.

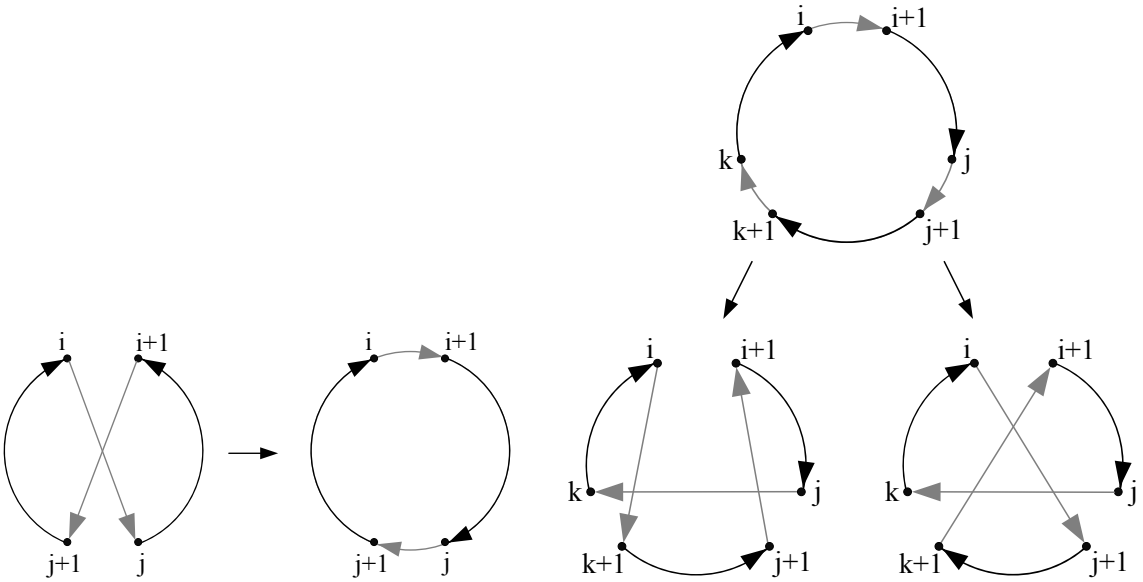


Figure 3.4: Local search procedures

The purpose of the last step, **acceptance phase**, is to determine whether to keep or discard the current solution. The acceptance phase exploits various strategies depending on the number of feasible solutions and their distribution in the solution space. For example, the first improved strategy used in this thesis accepts only solutions with a better cost. While exploration of the solution space is still supported due to the random components of the neighborhood operators, this strategy focuses more on exploiting. It allows us to achieve significant improvements in short time.

Another commonly used acceptance strategy is the simulated annealing by Kirkpatrick et al. [47]. Along with improved solutions, it also accepts non-improved solutions with a certain probability, which decreases over time. Our preliminary studies show that the simulated annealing does not give significant improvements for our real-life scenarios within the given 5-minutes time slot and, therefore, the first improved strategy was preferred.

3.3 Solving CMP with Inter-Depot Routes

The CMPID differs from the CMP in one major aspect - it allows vehicles to travel between the stations. This limits applicability of the CMP methods to this problem and affects their performance.

The Clarke and Wright algorithm as well as the proposed modified version has a high computational speed and simplicity, but lacks flexibility [23]. It cannot be easily

adapted to the inter-depot extension and is therefore discarded, regardless of its good performance on the CMP instances.

The idea behind insertion heuristics is very general: they iteratively extend a solution, one point at a time, based on a metric. In order to adapt an insertion heuristic to a new constraint, it is sufficient in many cases to change only its metric.

In this section, two new insertion heuristics for the CMPID are introduced: inter-depot insertion heuristic (IDIH) in Section 3.3.1 and IDIH with battery reservations (IDIH-Reserve) in Section 3.3.2. Some details on the VNS for the CMPID are given in Section 3.3.3.

3.3.1 Inter-Depot Insertion Heuristic

The inter-depot insertion heuristic does not exploit the queue principle from the queue-based insertion heuristic (see Section 3.2.2). Instead, the IDIH selects both a vehicle and a point for insertion based on an evaluation function. This enables the exploration of more point-vehicle combinations.

The aforementioned evaluation function selects a point-vehicle pair based on three parameters: traveling distance from the vehicle's position to the point, last visit time of the point, and expected arrival time. The first parameter aims at minimizing the total travel distance of a vehicle. Minimizing the last visit time directs the search towards the points that have not been visited the longest. Finally, minimizing the arrival time algorithm indirectly minimizes time delays between visits of the points. The evaluation function combines these parameters as follows:

$$\begin{aligned} g'(p,v) =& \alpha_1 \cdot d(v,p) + \alpha_2 \cdot \tau_p + \alpha_3 \cdot art(v,p) \ , \\ & \alpha_1 + \alpha_2 + \alpha_3 = 1, \alpha_1 \geq 0, \alpha_2 \geq 0, \alpha_3 \geq 0 \ , \end{aligned}$$

where p and v are a point and a vehicle considered for insertion, $d(v,p)$ is the distance from the last point visited by vehicle v to point p, τ_p is the last visit time of the point, $art(v,p)$ is the expected arrival time of vehicle v at point p. The arrival time is measured from the beginning of the mission and takes into account the current routes of vehicle v. Coefficients α_1, α_2 and α_3 dictate the influence of each parameter. Section 4.2.2 suggests the best values of the coefficients based on performed evaluations.

The workflow of the IDIH is presented in Algorithm 3. First of all, a route is created for every vehicle v by adding its given initial location $inLoc_v$ as the first element. Then the main loop in lines 3–35 inserts new points to the solution, one at a time. It terminates when no point-vehicle pair is selected (line 29).

Algorithm 3: Inter-depot insertion heuristic

input : problem description
output: a set of routes $R = \{R_v | v \in V\}$

$R \leftarrow \{\}$;
for $v \in V$ **do** $R_{v,1}$.ADD($inLoc_v$) ;
repeat
  $bestVehicle \leftarrow null$;
  $bestPoint \leftarrow null$;
  $bestChStation \leftarrow null$;
  $minValue \leftarrow \infty$;
  for $v \in V$ **do**
    for $p \in N_p$ **s.t.** NOMTIMEVIOLATION(R_v, p) **and** $p \neq$ LASTPOINT(v) **do**
      if NEEDBATCHANGE(v,R_v,p) **then**
        $chStation \leftarrow$ COMPUTESTATION(v, R_v, p);
        if $chStation \neq null$ **and** NOCOLLISION($v, R_v, p, chStation$) **and** $g'(p, v) < minValue$ **then**
          $bestVehicle \leftarrow v$;
          $bestPoint \leftarrow p$;
          $bestChStation \leftarrow chStation$;
          $minValue \leftarrow g'(p, v)$;
        end
      else
        if NOCOLLISION($v, R_v, p, null$) **and** $g'(p, v) < minValue$ **then**
          $bestVehicle \leftarrow v$;
          $bestPoint \leftarrow p$;
          $bestChStation \leftarrow null$;
          $minValue \leftarrow g'(p, v)$;
        end
      end
    end
  end
  if $bestVehicle = null$ **then**
    break;
  end
  if $bestChStation \neq null$ **then**
    $R_{bestVehicle}$.INITIALIZENEWROUTE($bestChStation$);
  end
  $R_{bestVehicle,last}$.ADD($bestPoint$);
until *false*;
ADDFINALSTATIONS();
return R;

The algorithm first assigns initial values of the main variables: a vehicle $bestVehicle$ and a point $bestPoint$ for the insertion, their value $minValue$ of the evaluation function $g'(p, v)$ and an intermediate base station $bestChStation$, where vehicle $bestVehicle$ will renew its capacity if it is required (otherwise, $bestChStation$ is equal to $null$).

The algorithm selects a point-vehicle pair that has the minimal value of the evaluation function and does not violate the following constraints:

- after visiting point p, vehicle v can still land at a station before the end of mission time mt. It is ensured by function noMTimeViolation(R_v, p);
- point p was not the last point at the last route of vehicle v (the second condition in line 9), i.e., a vehicle cannot visit the same point again without visiting at least one other point or station;
- no other vehicle will visit the point at the same expected arrival time. This is indicated by function noCollision$(v, R_v, p, chStation)$ returning *true*. The last argument of this function is an intermediate station for changing a battery, if required;
- visiting point p does not cause violation of the battery capacity constraint.

The last condition is ensured by the following steps. The algorithm checks if vehicle v can visit point p and land at a station with its remaining on-board capacity. In case of success, function needBatChange(v,R_v,p) returns *false*. Otherwise, the vehicle must change its battery at an intermediate station before visiting the point. The intermediate station is chosen by function computeStation(v, R_v, p) described in Algorithm 4. The station must be reachable by vehicle v, have a spare battery of the corresponding type, and have the shortest distance to the insertion point and the current vehicle's location. If there is no such base station, this function returns *null*. This means that point p cannot be visited by vehicle v without violating the capacity constraint, and, therefore, this point-vehicle pair is not considered as a candidate for insertion.

3.3.2 Inter-Depot Insertion Heuristic with Reservations

The IDIH with battery reservations (IDIH-Reserve) extends the IDIH with a battery reservation procedure that includes a negotiation mechanism. The purpose of this mechanism is to distribute batteries among vehicles as evenly as possible. It enables the simultaneous use of the maximal number of vehicles and, as a consequence, achieves a better visitation frequency.

Algorithm 4: Function computeStation(v, R_v, p)

```
1 bestStation ← null;
2 minDist ← ∞;
3 for b ∈ N_b do
4     if d_{v,b} + d_{b,p} < minDist and nBat_{b,type_v} > 0 and canReach(R_v, b) then
5         minDist ← d_{v,b} + d_{b,p};
6         bestStation ← b;
7     end
8 end
9 return bestStation;
```

Let us now explain the basic steps of the IDIH-Reserve. These steps are based on the assumption that the available batteries are sufficient for all vehicles to fly until the end of the mission. The steps needed if this assumption does not hold are explained afterwards.

The basic steps of the IDIH-Reserve heuristic are presented in Algorithm 5. As the IDIH, it first initializes the routes of the vehicles by inserting their initial locations. Then the heuristic iteratively adds one point at a time to the last route of a vehicle. Both a point and a vehicle are selected based on the evaluation function so that no constraints are violated. The iterations terminate when no more points can be added to a solution.

The differences between the IDIH-Reserve and the IDIH are the following:

1. The IDIH-Reserve introduces a battery reservation mechanism based on auctions. The goal of the reservation mechanism is to ensure that, at any time during the solving process, every vehicle has at least one reachable battery reserved for its next flight. It is implemented via the procedures reserveBatteries(v) for initial reservations, and updateReservations($bestVehicle, bestChStation$) for keeping the reservations up-to-date. More details about the procedures are given below.

2. The IDIH-Reserve extends the function needBatChange(v,R_v,p). This function determines if vehicle v must visit an intermediate station before traveling to point p. In the IDIH, the visit was required only if vehicle v could not reach a station after visiting point p. The IDIH-Reserve considers a second case, when the battery change is needed. It applies if a vehicle can reach some stations but cannot change its battery in any of them. This one-step lookahead proce-

Algorithm 5: Inter-depot insertion heuristic with battery reservations

input : problem description
output: a set of routes $R = \{R_v | v \in V\}$

$R \leftarrow \{\}$;
for $v \in V$ **do** $R_{v,1}$.ADD($inLoc_v$) ; RESERVEBATTERIES(v) ;
repeat
  $bestVehicle \leftarrow null$;
  $bestPoint \leftarrow null$;
  $bestChStation \leftarrow null$;
  $minValue \leftarrow \infty$;
  for $v \in V$ **do**
    for $p \in N_p$ **s.t.** NOMTIMEVIOLATION(R_v, p) **and** $p \neq$ LASTPOINT(v) **do**
      if NEEDBATCHANGE(v,R_v,p) **then**
        $chStation \leftarrow$ COMPUTESTATION(v, R_v, p);
        if $chStation \neq null$ **and** NOCOLLISION($v, R_v, p, chStation$) **and** $g''(p, v) < minValue$ **then**
          $bestVehicle \leftarrow v$;
          $bestPoint \leftarrow p$;
          $bestChStation \leftarrow chStation$;
          $minValue \leftarrow g''(p, v)$;
        end
      else
        if NOCOLLISION($v, R_v, p, null$) **and** $g''(p, v) < minValue$ **then**
          $bestVehicle \leftarrow v$;
          $bestPoint \leftarrow p$;
          $bestChStation \leftarrow null$;
          $minValue \leftarrow g''(p, v)$;
        end
      end
    end
  end
  if $bestVehicle = null$ **then break** ;
  if $bestChStation \neq null$ **then**
    $R_{bestVehicle}$.INITIALIZENEWROUTE($bestChStation$);
  end
  $R_{bestVehicle,last}$.ADD($bestPoint$);
  UPDATERESERVATIONS($bestVehicle, bestChStation$);
until *false*;
ADDFINALSTATIONS();
return R;

dure helps to avoid the situation in which a drone has to terminate its mission prematurely, as it can only reach stations without energy supplies.

3. Another modified function is COMPUTESTATION(v, R_v, p). This function selects an intermediate base station for vehicle v to renew its capacity. In the IDIH, the chosen intermediate station must be reachable by vehicle v and have the minimal distance to both point p and the last point visited by v. In contrast to the IDIH, the IDIH-Reserve prefers the base stations that have the shortest distance only to point p. This selection criteria saves the maximal amount of energy for the next route, while still fulfilling the capacity constraint.

4. The evaluation function of the IDIH-Reserve is based on relative time-based parameters instead of absolute ones. The function is discussed in detail later.

The main steps of the reservation procedure RESERVEBATTERIES(v) are depicted in Algorithm 6. In lines 1–6, vehicle v is assigned one battery from each station that 1) is reachable by v and 2) has a non-reserved battery of the corresponding type. The reservations $resBatIn_v$ of vehicle v are defined as a set of stations that fulfill the aforementioned constraints. If vehicle v could not reserve any battery at any station, it would not be able to continue its mission after it uses all capacity on board. Therefore, to receive a battery for the next flight, vehicle v initiates an auction involving vehicles of the same type. During the auction such vehicles offer every reserved battery for which the following conditions are fulfilled. First of all, an offering vehicle must have more than one reserved battery. Secondly, it offers every battery that is in a station that the initiating vehicle v can reach. Then vehicle v selects the best offer, which is a battery located at the closest station. In case of ties, offering vehicles with more reserved batteries are preferred.

After the initialization steps, the two *for*-loops of the IDIH-Reserve iterate through all possible point-vehicle pairs to select a feasible combination with the minimum evaluation value. The selected pair must fulfill the same set of constraints as in the IDIH with the aforementioned enhancements.

If a point-vehicle pair was selected, the heuristic performs insertion and updates battery reservations for the selected vehicle. The reservations are updated by using the procedure UPDATERESERVATIONS$(bestVehicle, bestChStation)$. This procedure is similar to RESERVEBATTERIES(v) with the only difference that all previous reservations of $bestVehicle$ must be canceled before making new ones. Since vehicle $bestVehicle$ will use a battery from station $bestChStation$, one battery is removed from this station.

Algorithm 6: Negotiation-based function RESERVEBATTERIES

input : vehicle v
output: updated reservations $\forall v' \in V \;\; resBatIn_{v'}$

for $b \in N_b, \;\; nBat_{b,type_v} > 0$ **do**
  if $d_{inLoc_v,b}/speed_{type_v} \leq inRemCap_v$ **then**
    $resBatIn_v$.ADD(b) ;
    $nBat_{b,type_v} \leftarrow nBat_{b,type_v} - 1$;
  end
end
if $|resBatIn_v| = 0$ **then**
  $bestVehicle \leftarrow null$;
  $bestStation \leftarrow null$;
  for $v' \in V, \;\; type_{v'} = type_v, \;\; v' \neq v, |resBatIn_{v'} > 1|$ **do**
    for $b' \in resBatIn_{v'}$ **do**
      if $d_{inLoc_v,b'}/speed_{type_v} \leq inRemCap_v$ **then**
        if $bestVehicle = null$ **then**
          $bestVehicle \leftarrow v'$;
          $bestStation \leftarrow b'$;
        else
          if $(d_{inLoc_v,b'} < d_{inLoc_v,bestStation})$ **or**
          $(d_{inLoc_v,b'} = d_{inLoc_v,bestStation}$ **and**
          $resBatIn_{v'} > resBatIn_{bestVehicle})$ **then**
            $bestVehicle \leftarrow v'$;
            $bestStation \leftarrow b'$;
          end
        end
      end
    end
  end
  if $bestVehicle \neq null$ **then**
    $resBatIn_{bestVehicle}$.REMOVE($bestStation$);
    $resBatIn_v$.ADD($bestStation$);
  end
end

An evaluation function, as mentioned before, reflects the optimization criteria but is not equal to it. The evaluation function of the IDIH-Reserve selects a point and a vehicle to be inserted, based on the three parameters:

- distance from the current position of a vehicle to a point. This parameter minimizes the total travel distance;
- arrival time of a vehicle at a point. When this parameter is minimized, a vehicle with the shortest duration of the current schedule is selected;
- time of the last visit at a point in order to select the longest waiting point.

Every time routes are extended, time-based parameters increase. On the contrary, the distance parameter always stays within boundaries, because the distance matrix is fixed. The influence of such controversial behavior on the selection process should be minimized. Therefore, the evaluation function of the IDIH-Reserve uses relative time-based parameters, i.e., parameters based on other variable measurements. Relative arrival time $\Delta art(v,p)$ is computed as $art(v,p) - min_{v' \in V}\,[cmt_{v'}]$, the difference between the actual arrival time $art(v,p)$ and the minimal current mission time of a vehicle (see Figure 3.5). Relative time of the last visit $\Delta\tau_p$ equals to $\tau_p - min_{p' \in P}\,[\tau_{p'}]$, the difference between the last visit time τ_p and the minimal last visit time among all points (see Figure 3.6).

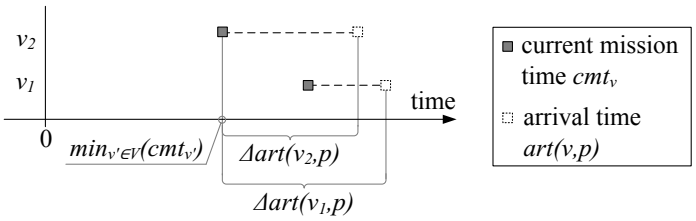


Figure 3.5: Illustration of the relative arrival time $\Delta art(v,p)$ for vehicles v_1 and v_2.

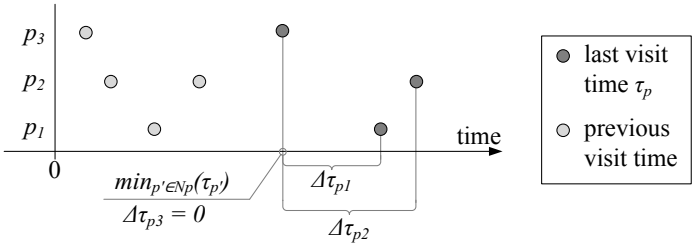


Figure 3.6: Illustration of the relative last visit time $\Delta\tau_p$ for vehicles v_1 and v_2.

Taking into account the aforementioned changes, the evaluation function of the IDIH-Reserve is computed according to the following formula:

$$g''(p,v) = \alpha_1 \cdot d(v,p) + \alpha_2 \cdot \Delta art(v,p) + \alpha_3 \cdot \Delta\tau_p \ ; \qquad (3.1)$$
$$\alpha_1, \alpha_2, \alpha_3 \geq 0; \ \ \alpha_1 + \alpha_2 + \alpha_3 = 1 \ ,$$

where $d(v,p)$ is the distance that vehicle v has to travel to point p, $\Delta art(v,p)$ is the relative arrival time of vehicle v at point p, $\Delta\tau_p$ is the relative last visit time of point p. Weight coefficients α_1, α_2 and α_3 define which of the mentioned parameters influence the final decision more. The coefficient *scale* used in the IDIH is not required, as all parameters have the same order of magnitude. Section 4.2.2 suggests the best performing values of the coefficients.

The steps described above are sufficient to solve the CMPID problem with a large number of charged batteries. However, it is not as efficient if not all drones can fly simultaneously throughout the mission, due to the small number of batteries. Next we describe how the IDIH-Reserve heuristic is extended to consider this case.

Monitoring tasks require that up-to-date information is provided throughout the mission time. Therefore, in case of very limited energy resources, they should be distributed among the vehicles so that the maximum number of them covers the whole planning horizon. Such distribution is illustrated in Fig. 3.7. The figure depicts an example in which two vehicles have to monitor three target points with four batteries. The left image represents the battery distribution that can be achieved with the basic steps. The image on the right demonstrates the optimal battery distribution: vehicle 1 utilizes all four batteries, whereas vehicle 2 is not deployed. This is the aim of an extended version of the heuristic. In addition to covering the whole mission time, minimizing the number of deployed vehicles has an additional advantage: it increases safety especially for a challenging environment.The top charts show assignments of the batteries to the vehicles. The bottom charts demonstrate the resultant distribution of visits. Red lines indicate given mission time.

The second strategy can be extended further to consider cases when a vehicle has not enough batteries to cover the whole mission time, e.g., in the aforementioned example, vehicle 1 has only 3 batteries. For this purpose, a vehicle can stay at the base stations longer than needed to renew its capacity. Such waiting time can be distributed evenly between visits to the stations and computed as the remaining time to cover, i.e., the difference between the mission time and the maximal possible mission time given the number of batteries, divided by the number of battery changes.

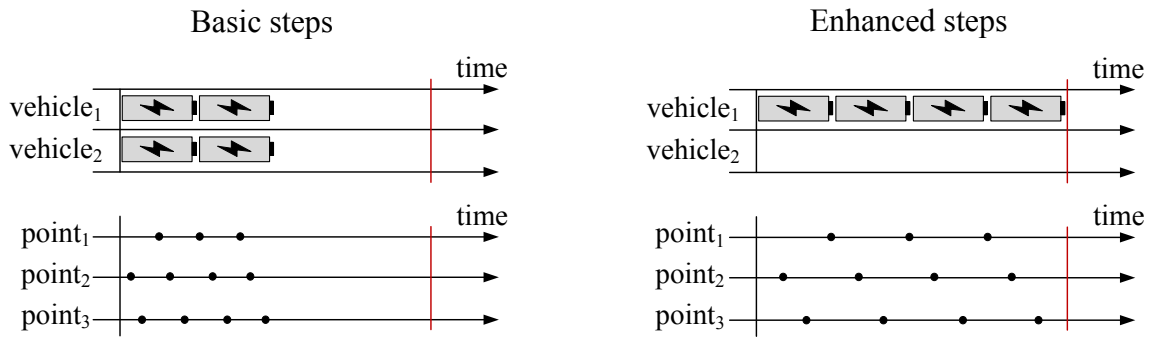


Figure 3.7: Solution parameters for the IDIH-Reserve with the basic and enhanced steps.

The extended IDIH-Reserve regulates distribution of the batteries according to the following steps:

1. Estimate the number of batteries required for a UAV of each type to cover the whole mission time:

$$nBatMT_t = \left\lceil \frac{mt - avgRemCap_t}{batCap_t + tChBat_t} \right\rceil \ ,$$

where mt is the given mission time, $avgRemCap_t$ is the average initial remaining capacity of vehicles of type t, $batCap_t$ is the battery capacity of this type, $tChBat_t$ is the time required to change a battery.

2. Based on this estimation, compute the maximum number of vehicles of type t that can fly until the mission ends:

$$maxNVeh_t = \left\lfloor nBat_t / nBatMT_t \right\rfloor \ ,$$

where $nBat_t$ is the total number of batteries of this type. In case $maxNVeh_t < 1$, the time $tChBat_t$ is increased s.t. $maxNVeh_t = 1$, i.e., after changing its battery a drone of type t will wait at the ground station in order to cover the whole mission time.

3. Estimate the number of batteries that remain after all $maxNVeh_t$ vehicles utilize their batteries:

$$nBatRemain_t = nBat_t - nBatMT_t \cdot maxNVeh_t \ .$$

4. Start performing the basic steps of the IDIH-Reserve algorithm. As soon as $maxNVeh_t$ vehicles of type t start their second route, an additional constraint

is imposed for vehicles of this type. Only these $maxNVeh_t$ vehicles are allowed to change their batteries as long as required. If some batteries still remain ($nBatRemain_t \neq 0$) then one more vehicle can use $nBatRemain_t$ batteries. Other vehicles terminate their mission after finishing their first routes.

3.3.3 Variable Neighborhood Search

Within this implementation of the variable neighborhood search, an initial solution is constructed by the IDIH or IDIH-Reserve. The other steps of the VNS for the CMPID are discussed further.

In the **shaking step** a solution is modified by an operator in order to switch to another solution in the neighborhood. The new operator *replace* is introduced and applied together with the three operators used for the CMP problem: the *insert* [54], *move*, and *cross-exchange* operators [41] (see Section 3.2.3). The *move(x, η)* operator relocates a sequence of η points from one route to another. The *cross-exchange(x,η)* operator exchanges two sequences of η points between two different routes. The *insert(x,η)* operator adds η new points, each at some position in a chosen route. Finally, the *replace(x,η)* operator substitutes η routed points with new points that have least number of visits. All sequences, points and routes are selected randomly. These operators are applied in the following order: *insert(x,η)*, *replace(x,η)*, *move(x,η)*, *exchange(x,η)* with $\eta = 1$. The selection of both order and value of η is discussed in Section 4.2.2.

Local search step performs only local optimization. For example, it can minimize the travel distance of each route modified during the shaking step. The VNS for the CMPID is based on the reduced variable neighborhood search, a variation of the VNS without local search [37]. Due to the inter-depot routes, complexity of the problems increased comparing to the CMP. Therefore, the local search requires larger computational effort while delivering smaller improvements.

Acceptance phase determines whether the new solution provided by the local search is accepted or not. This phase utilizes the same strategy as the VNS for the CMP. It is called the first improvement strategy, because it selects every solution that has a better value of the goal function.

3.4 Solving CMPID with Priorities

The objective of the CMPIDP is different to the CMP and CMPID problems solved in the previous sections. The CMP and CMPID aim at distributing visits evenly over

the fixed mission time. The CMPIDP does not include the fixed mission time. Thus, by minimizing the time lag between the visits, it minimizes the overall mission time as well. Solution of the CMPIDP exploits all vehicles simultaneously to perform visits with shorter time delays between visits. With the same number of charged batteries, visits of a CMPID solution can be allocated farther from each other in order to cover a fixed planning horizon.

Another distinction between the CMPIDP and the CMP/CMPID is the presence of priorities. Priority of a point defines its importance and, as a consequence, the ratio of its visit frequency as well as the visit frequency of the points with other priority values.

An unlimited planning horizon makes the search space wider, whereas the priorities lead to a more constrained problem. The restricted computational time as in the CMP and CMPID and the increased complexity prevent the use of metaheuristics, which produce improvements but are too time-consuming. Therefore, in this section we focus on a construction heuristic. In case when the longer computational time is acceptable, the VNS described in the previous section can be used.

3.4.1 Inter-Depot Insertion Heuristic with Reservations

This algorithm is the IDIH-Reserve heuristic adapted to the CMPIDP problem. This was realized in the following two steps: eliminating the information about the fixed mission time, and incorporating the priority-related information. The first step is realized by utilizing only the basic steps without the battery redistribution and the noMTimeViolation(R_v, p) function (see Algorithm 2). During the second step the evaluation function $g''(p, v)$ (3.1) was modified as follows:

$$g'''_{pr}(p, v) = \alpha_1 \cdot d(v, p) + \alpha_2 \cdot \Delta art(v, p) + \alpha_3 \cdot \Delta\tau_p/(pr_p)^\beta + \alpha_4 \cdot scale \cdot nVis_p/(pr_p)^\beta;$$
$$\alpha_1, \alpha_2, \alpha_3, \alpha_4, \beta \geq 0 \ ; \ \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \ , \tag{3.2}$$

where $d(v, p)$ is the distance that vehicle v has to travel to visit point p, $\Delta art(v, p)$ is the relative arrival time of vehicle v at point p, $\Delta\tau_p$ is the relative last visit time of point p, pr_p is the priority of point p, $nVis_p$ is the number of visits made at point p. More information on the relative coefficients is given in Section 3.3.2. Coefficient *scale* brings the number of visits to the same order of magnitude as other parameters. It was set to 100 for our scenarios. Coefficient β defines the influence of priorities. The bigger the β, the more points with higher priorities are preferred. The suggested values of all coefficients are reported in Section 4.3.2.

3.5 Constraint-based Programming

In order to obtain optimal solutions, the continuous monitoring problem and its extensions were modeled as constraint optimization problems and solved by a constraint programming solver. Further we describe the background terminology, and illustrate modeling techniques on the traveling salesman problem. Then we discuss several constraints from the models of the CMP, CMPID and CMPIDP.

A constraint satisfaction problem (CSP), also called a constraint network, is a mathematical problem defined by a tuple (X, D, C), where $X = \{x_1, x_2, ..., x_n\}$ is the set of variables, $D = \{d_1, d_2, ..., d_n\}$ is the set of domains such that $d_i \in D$ is a finite set of all possible values for variable $x_i \in X$ and $C = \{c_1, c_2, ..., c_m\}$ is a set of constraints.

An assignment sets values from domains to variables. An assignment is complete if every variable has an assigned value. An assignment is consistent if it satisfies all constraints. A *solution* of a constraint satisfaction problem is a complete consistent assignment.

Constraint optimization problem (COP) is a constraint satisfaction problem augmented with a goal function. A solution of a COP is a complete consistent assignment of values to variables that minimizes/maximizes the goal function.

Modeling of COPs is performed in a declarative way, i.e., it requires only a definition of solution properties in a form of constraints without specifying solving steps. To solve the obtained model, a CP solver uses various search techniques like constraint propagation and backtracking. More information on constraint processing techniques can be found in [26].

Constraint optimization problems can be modeled with a constraint programming (CP) language such as MiniZinc [63] and then solved by a CP solver. MiniZinc is a declarative modeling language that allows to model both CSP and COP. This language has a number of advantages. Due to its simplicity and expressivity, MiniZinc is suitable for fast prototyping. A MiniZinc program consists of three major parts: parameters and variable declarations, constraints as well as a solve statement. The language supports definition of predicates and provides a set of global constraints. Another advantage is that a MiniZinc model can be easily transformed to a FlatZinc model which can be solved by many existing CP solvers, e.g., Gecode, Opturion CPX, OR-Tools or Choco. In this thesis the CP solver Gecode [35] has been used as it has won a number of the recent MiniZinc competitions in 2008–2012.

Let us demonstrate the modeling techniques of the MiniZinc on a sample bin packing problem (BPP) [34]. Given a set of objects, each with a predefined size, and a set of bins with a limited capacity, the goal is to pack the objects into the minimal number of bins.

The aforementioned input parameters can be defined in MiniZinc as shown in Listing 3.1. Declaration of a parameter includes its type. MiniZinc supports such basic types as integers (int), booleans (bool), floating point numbers (float), and strings (string) as well as two composite types, arrays and sets. An array or a set includes values of the same basic type. For example, the array size contains size of every object that is integer. To declare an array, one must specify numbering of every dimension and the type of its elements. The array size is one dimensional, and the number of its elements equals nObjects, i.e., [1.. nObjects] defines the numbering.

```
1 int: nObjects;                          % number of objects
2 array[1..nObjects] of int: size;        % the size of each object
3 int: binSize;                           % size of a bin
4 int: nBins;                             % number of available bins
```

Listing 3.1: Declaration of the input parameters of the bin packing problem

Let us consider an example of the BPP instance. Three objects should be packed into at most three bins. The sizes of the objects are 5, 10 and 3. The maximum load of a bin is equal to 10. The listing below defines the described input in the MiniZinc format.

```
1 nObjects = 3;    size = [5,10,3];    binSize = 10;    nBins = 3;
```

Listing 3.2: An example of the bin packing problem instance

Next we define the decision variables that represent a solution, see Listing 3.3. The decision variables are declared with their domain and a keyword var. A domain can be specified either by the type of a variable or by a range (e.g., [1..10]). The array packed reflects assignment of objects to bins. If object o is loaded into bin b, then the element packed[b,o] equals 1; and 0 otherwise. Note that the elements of this array have the domain defined as a range [0..1] . The variable binLoad[b] is the total size of objects packed into bin b. Its value cannot exceed the size of a bin, what is ensured by its domain. The number of bins loaded with objects is defined as variable nUsedBins. Its value must be minimized.

```
1 array[1..nBins, 1..nObjects] of var 0..1: packed;
2 array[1..nBins] of var 0..binSize: binLoad;
3 var 0..nBins: nUsedBins;
```

Listing 3.3: Declaration of the decision variables of the bin packing problem

Constraints in MiniZinc are declared by a keyword constraint and a boolean expression. A simple boolean expression can be, for example, a relational expression such as a > b. More complex expressions consist of simple expressions connected with a boolean operator such as conjunction (/\), disjunction (\/), negation (not) and implication (->). Keywords forall and exists are used to generate multiple conjunctions and disjunctions, respectively.

Listing 3.4 demonstrates how the constraints of the BPP can be represented. The total load of every bin is computed by a built-in aggregation function sum in lines 2–4. The summands are defined by the expression size[o]*packed[b,o], where o and b are replaced with elements of the sets {1..nObjects} and {1..nBins}, respectively. The next statement in lines 6–8 guarantees that every object is packed in exactly one bin. The equality in the last line counts the number of loaded bins. It uses a built-in function bool2int that translates boolean values to integers: true is interpreted as 1, and false as 0. Thus, the aggregate function sum counts the number of bins that fulfill the condition binLoad[b] > 0.

```
1 constraint
2     forall(b in 1..nBins) (
3        binLoad[b] = sum(o in 1..nObjects) (size[o]*packed[b,o]) )
4     /\
5     forall(o in 1..nObjects) (
6        sum(b in 1..nBins) (packed[b,o]) = 1 )
7     /\
8     nUsedBins = sum(b in 1..nBins) (bool2int(binLoad[b] > 0));
```

Listing 3.4: The constraints of the bin packing problem

The optimization objective of the classical BPP is to minimize the number of bins as in Listing 3.5. The MiniZinc allows both minimization (keyword minimize) and maximization (maximize) objectives and guarantees finding an optimal solution. To solve a decision problem, the keyword satisfy should be used instead.

```
1 solve minimize nUsedBins;
```

Listing 3.5: The objective function of the BPP

Listing 3.6 summarizes the complete model of the bin packing problem. The last line defines the variables that should be displayed in the output of the solver, i.e., nUsedBins and packed.

The remainder of the section describes several constraints of the CMP model. The complete encodings of the CMP, CMPID and CMPIDP as well as an example of the instances are given in Appendixes A and B.

```
% INPUT
int: nObjects;                          % number of objects
array[1..nObjects] of int: size;  % the size of each object
int: binSize;                           % size of a bin
int: nBins;                             % number of available bins


% OUTPUT (decision variables)
% 1 if an object is packed in a bin; 0 otherwise
array[1..nBins, 1..nObjects] of var 0..1: packed;

% the total size of objects packed into a bin
array[1..nBins] of var 0..binSize: binLoad;

% number of used bins
var 0..nBins: nUsedBins;

% CONSTRAINTS
constraint
  % compute the total size of the objects in a bin;
  % it cannot exceed the size of a bin - it is ensured by the domain
     forall(b in 1..nBins) (
        binLoad[b] = sum(o in 1..nObjects) (size[o]*packed[b,o])
     )
     /\ % every object is packed exactly once
     forall(o in 1..nObjects) (
        sum(b in 1..nBins) (packed[b,o])  = 1
     )

     /\ % calculate the number of used bins
     nUsedBins = sum(b in 1..nBins) (bool2int(binLoad[b] > 0));

% minimize the number of used bins
solve minimize nUsedBins;

% PRINT
output [ show(nUsedBins), show(packed)];
```

Listing 3.6: The complete model of the bin packing problem

Let us first introduce the input parameters and decision variables required for defining the constraints. The parameters are shown in Listing 3.7. It contains two new terms maxNVisitsV and maxNVisitsP that are used for declaring the decision variables. The maxNVisitsV is the maximal number of nodes that a vehicle can visit including its initial location. The parameter maxNVisitsP estimates the maximal number of observations taken at a point.

```
int: nSt;              % number of stations
int: nP;               % number of points
int: nN = nSt + nP;    % number of nodes
int: nT;               % number of types

```

```
% number of batteries of a type in a station
array[1..nSt, 1..nT] of int: nBat;

int: nV;                                % number of vehicles
array[1..nV] of 1..nT: vehType;         % type of a vehicle
array[1..nV] of 1..nN: initLoc;         % initial location of a vehicle
array[1..nV] of 1..nSt: vehStation; % home station of a vehicle

int: maxNVisitsV; % maximal number of visits made by a vehicle
int: maxNVisitsP; % maximal number of visits of a point
int: mT;          % misssion time
```

Listing 3.7: A subset of the input parameters from the CMP model

A solution of the CMP is a set of routes for every vehicle. We model it as an array of variables visit (Listing 3.8), where every element visit[v,j] is a j-th node that vehicle v visits. Nodes are represented as integer numbers: 1..nSt for stations and nSt+1..nN for points. All routes of one vehicle are defined as a sequence of nodes.

```
array[1..nV, 1..maxNVisitsV] of var 1..nN+1: visit;
array[1..nP,1..maxNVisitsP] of var int:aTimeP;
array[1..nP] of var int: firstArTime;
array[1..nP] of var int: costs;
```

Listing 3.8: A subset of the decision variables from the CMP model

Vehicles might visit different number of nodes, while their routes are represented by an array with fixed length maxNVisitsV. Therefore, this model introduces the notion of a "dummy" node that is equal to nN+1 and determines the end of the last route. In other words, all array elements after the last visited node equal to the dummy node. For example, there are 4 points to be visited and a station. A vehicle can visit at most 10 nodes, i.e., maxNVisitsV=10, and its route is $1-2-3-4-1-5-1$. Then these routes can be represented as the following array: visit $= [1, 2, 3, 4, 1, 5, 1, 6, 6, 6]$, where the dummy node equals 6. We have selected the value nN+1 for a dummy node instead of 0 because of the admissible indexes of distance matrix that are positive integers that exclude 0.

The other decision variables are used to compute a value of the goal function. The variable aTimeP is the sequence of arrival times for each point, ordered in decreasing order. In case if a point has less visits than maxNVisitsP, the remaining array elements are set to 0. The variable firstArTime contains the first arrival time of every point. The penalty cost of every point is stored in costs.

Listing 3.9 describes several constraints of the CMP. The first constraint ensures that every vehicle starts its first route at the initial location. The constraint in lines 3–8 allows vehicles to visit only their home stations vehStation and picture points, i.e., nodes from the range nSt+1..nN. The last constraint guarantees that the vehicles do

not use more batteries than there are available. It computes the number of times when vehicles of type iT changed the batteries at station iS, i.e., number of visits at this station excluding the last visit when a UAV lands and the first visit when the station can be the initial location of a vehicle.

```
1  constraint forall(iV in 1..nV)(visit[iV,1] == initLoc[iV]);
2
3  constraint
4    forall(iV in 1..nV)(
5      forall(iM in 2..maxNVisitsV)(
6        visit[iV,iM] = vehStation[iV] \/ visit[iV,iM]>nSt ) );
7
8  constraint
9    forall(iS in 1..nSt)(
10     forall(iT in 1..nT)(
11       let{var int: nChanges = sum(iV in 1..nV, iVis in 2..maxNVisitsV-1)
12         (bool2int(vehType[iV] == iT /\ visit[iV,iVis] == iS /\
13           visit[iV,iVis+1] != nN+1)) } in
14       nChanges <= nBat[iS, iT]
15     ) );
```

Listing 3.9: A subset of the constraints from the CMP model

The continuous monitoring problem minimizes the squared delays between the consecutive visits of all points. Its objective function (2.1) can be modeled as presented in Listing 3.10. Previously we defined a decision variable costs in a form of array, where every element is the penalty for a single point, i.e., a sum of the squared delays. The constraint in the listing defines these penalties for two possible cases: if there are no observations of point `iP` (lines 3–4) and if the point was visited at least once (lines 6–13). If a point was not visited, the cost of the point penalizes the whole mission time and the parameter lvt_p, i.e., the difference between the last observation before the monitoring started and the beginning of the mission. In the second case when the point was observed, the penalty consists of three parts: a penalty on the time before the first visit (line 13), a penalty on the delays between the visits (lines 7–10) and a penalty on the time from the last visit to the end of the mission (line 12). The total cost is derived in line 16. The statements in line 17 specify the optimization objective.

The constraint model partially presented in this section allows to calculate the optimal solutions for the CMP, CMPID, and CMPIDP problems. Although solving the model with a CP solver finds the optimal solution, its applicability is limited to instances with only a few points. The real-life scenarios contain hundreds of points and, thus, can only be solved by an approximate algorithm. Comparisons of the

heuristic approaches proposed in the previous sections with optimum are presented in Sections 4.1.3, 4.2.3, and 4.3.3.

```
constraint
  forall(iP in 1..nP)(
    aTimeP[iP, maxNVisitsP] == 0 /\
    costs[iP] = (mT + lvt[iP])*(mT + lvt[iP])
    \/

    aTimeP[iP, maxNVisitsP] != 0 /\
    let{var int: costBetweenVisits = sum(i in 1..maxNVisitsP-1)
      ((aTimeP[iP,i+1] - aTimeP[iP,i]) *
       (aTimeP[iP,i+1] - aTimeP[iP,i]) *
       bool2int(aTimeP[iP, i] !=0))} in

    costs[iP] = costBetweenVisits +
      (mT - aTimeP[iP,maxNVisitsP])*(mT - aTimeP[iP,maxNVisitsP])+
      (firstArTime[iP] + lvt[iP])*(firstArTime[iP] + lvt[iP])
  );

var int: totalCost = sum(iP in 1..nP)(costs[iP]);
solve minimize totalCost;
```

Listing 3.10: Calculation of the goal function value

3.6 Incorporating Environmental Changes

Often route planning algorithms are designed for stable conditions and do not consider possible environmental changes like failures of vehicles or changes of objectives. This section shows how IDIH-Reserve can be applied in case of changing conditions.

When the problem instance is changed, the mission must be re-planned accordingly. Let ΔT_R be the upper bound for the time needed to compute a new solution. For our real world scenarios this upper bound can be assigned to 15 seconds.

The actual re-planning is performed as follows. First, the UAVs receive a command to halt at the next node in their routes. If a UAV cannot reach this node within ΔT_R it halts at a location where it will arrive in ΔT_R. For such locations, "dummy" stations are created. They act only as departure nodes, do not store any batteries, and the drones cannot land there at the end of the mission. Next, a new problem instance is generated for the system state expected in ΔT_R. Based on the new problem instance, the mission is recomputed and sent to the vehicles. Ideally ΔT_R would be 0. However, if ΔT_R is significantly smaller than the travel times between points (which is the case for our method) then the impact of $\Delta T_R > 0$ on the solution quality is negligible.

The most common events and the corresponding updates of a problem instance are the following:

- *A UAV fails and can no longer operate.* Remove this UAV from the list, refresh the battery reservations.
- *An obtained picture is of poor quality and cannot be used.* Change its last visit time to the previous one.
- *There are new drones or batteries.* Add new drones/batteries, reassign the batteries.
- *There are new obstacles.* Recompute the picture points [72] and the distance matrix, set the last visit time of the points as described in Section **??**.
- *Priorities change.* Set new priority values.
- *A base station has moved.* Change location of the station, recompute the distance matrix and refresh the battery reservations. After moving the station some drones might not have enough capacity to reach any station. Then there is no feasible solution and the user should be warned.
- *A base station can no longer operate.* Remove the station from the list, recompute the distance matrix and refresh the battery reservations. If a drone cannot reach any station, no feasible solution exists. The situation described in the previous paragraph, where the drones are not able to land, might occur during this event. The corresponding warning should be delivered to a user.

3.7 Summary

The continuous monitoring problem and its extensions are variations of the vehicle routing problem that is NP-hard. Due to complexity of the CMP, CMPID and CMPIDP, exact approaches are currently not able to solve real-life instances. Therefore, we propose a number of approximate algorithms for each problem.

We developed two construction heuristics for the CMP: queue-based insertion heuristic (QI) and modified Clarke and Wright algorithm (CW). The QI orders all points into a queue and iteratively inserts them in this order into a solution as long as insertions are possible. The heuristic selects a position where a point is inserted based on an evaluation function. The CW is initialized with single-point routes, one

for each point. Then the routes with the maximal savings value are connected. The procedure is repeated for every coverage of an area separately until all batteries are used. The solutions obtained with the QI and CW are then improved by variable neighborhood search (VNS).

The CMPID extends the CMP with inter-depot routes, i.e., vehicles can renew their energy capacity at any of the stations. To solve this problem, we propose inter-depot insertion heuristic (IDIH) and IDIH with battery reservations (IDIH-Reserve). Similar to the QI, these approaches extend partial solutions by one point at a time. In contrast to the QI, they also exploit their evaluation functions for selecting a point that will be inserted. The main differences between the IDIH and IDIH-Reserve are the parameters of the evaluation function and a negotiation-based mechanism introduced in the IDIH-Reserve for a better battery use. These approaches can be used in a combination with a VNS.

The third problem, CMPIDP, includes priority information for every point and does not fix the mission time. We show how the IDIH-Reserve can be applied to the CMPIDP with minor changes.

The advantage of approximate algorithms over the exact approaches is their short computational time. This allows to apply our heuristics for plan adaptation to several environmental changes such as drone failure or changed wind conditions.

When a new heuristic is introduced, it should be compared with an optimum on instances where optimal solutions are computable in admissible time. To obtain optimal solution costs for the CMP, CMPID and CMPIDP, we model the problems as constraint optimization problems and solve the models with a constraint programming solver.

Chapter 4

Computational Results

This thesis is dedicated to three route planning problems: the continuous monitoring problem (CMP), the CMP with inter-depot routes (CMPID), and the CMP with inter-depot routes and priorities (CMPIDP). Each problem is solved by the heuristic approaches. Their computational results are shown in this chapter.

The performance of the heuristics was evaluated empirically according to several criteria. The first criteria is the deviation of their solutions from optimum. It is essential, as heuristic approaches do not guarantee to find an optimal solution. Then we evaluate solution quality and scalability of the heuristics on real-life scenarios, as they cannot be solved optimally in a feasible amount of time due to the problem complexity. The scalability study shows how the performance of the heuristics evolve with the increasing instance size.

The conducted studies have the following details in common. The heuristics are implemented in Java 1.7. All tests were performed on Intel Core i5 2.50 GHz system with 8GB RAM running Windows 7. During computations no rounding of costs was performed to eliminate the bias introduced by rounding [61].

The chapter groups the performed studies into sections based on the problem: the CMP heuristics in Section 4.1, the CMPID approaches in Section 4.2, and the CMPIDP algorithms in Section 4.3. Each section consists of four parts describing: the used benchmarks[1] and their generation process (*Test instances*), the parameter selection for the heuristics (*Tuning*), the performance with respect to optimum (*Comparison with Optimum*), and the performance on real-life scenarios and other large benchmarks (*Evaluation on Real-Life Instances*).

[1]All benchmarks are available online: `http://uav.lakeside-labs.com/publications/test-data/planning/`

4.1 Continuous Monitoring Problem

4.1.1 Test Instances

Two benchmark sets were generated to evaluate the CMP heuristics: 9 *optimum* and 36 *real-life* instances. Instances of two sets significantly differ in size, since optimum cannot be computed for the large scenarios.

Optimum instances include 6 picture points and 2 base stations. Coordinates of the points were generated randomly in the interval [0, 40]. Every base station stores 1 drone and 2–4 batteries with capacity equal to 50 time units. The drones fly with the average speed equal to 1. Their service time is set to 1 time unit, whereas battery change requires 5 time units. The vehicles are initially located at randomly selected picture points. The initial remaining capacity of every vehicle is chosen randomly from the interval $[d_{min},\ 50 - d_{min}]$, where d_{min} is the minimal distance between a base station and its initial location. The mission time is limited by the longest possible mission of a drone, i.e., up to four battery capacities.

A set of **real-life benchmarks** represents several scenarios that we had, e.g., a territory at the fire-fighters drill in Wietersdorf or the area around the university of Klagenfurt. Generation of these instances was performed in several steps. First, the size of an area that can be covered by one picture is derived from the flight altitude and the camera resolution. Then the minimal number of points is spread equally over the area of interest. Every picture must have a certain overlap with at least three neighboring images. It is required for their stitching afterwards. More details on the picture points placement, stitching procedure and its requirements can be found in [72]. The number of points in this scenarios ranges from 46 to 441.

Every scenario has a fleet of 3, 6 or 9 UAVs and a set of 3–22 batteries equally distributed between the base stations. Parameters such as drone's velocity, maximal flight duration and service time are set to the real values of the drones used in our project, i.e., 2.5 m/s, 1200 s and 3 s, respectively.

Some of the large instances contain obstacles where drones are not allowed to fly, e.g., buildings or tall trees. The obstacles are denoted as polygons. In order to take obstacles into account, distance matrix is computed based on visibility graph that is widely used in different areas including robotics [13]. Our visibility graph consists of nodes, i.e., picture points, base stations and obstacle points, as well as edges connecting them. Edges connect two nodes only if the straight line between them does not intersect any edge of an obstacle. Then distance between the picture

points and base stations is computed as the length of the shortest path between them on the visibility graph.

The following naming template is applied to all mentioned instances. The first letter of the name stands for the instance type: "O" for the *optimum* and "L" for the *real-life* instances. The following numbers represent the number of points, base stations, batteries, and vehicles, respectively. For example, the instance depicted by Figure 4.1 is named as "L_442p_3d_22b_6v". This instance is a *real-life* scenario that includes 3 base stations, 22 batteries and 6 vehicles.

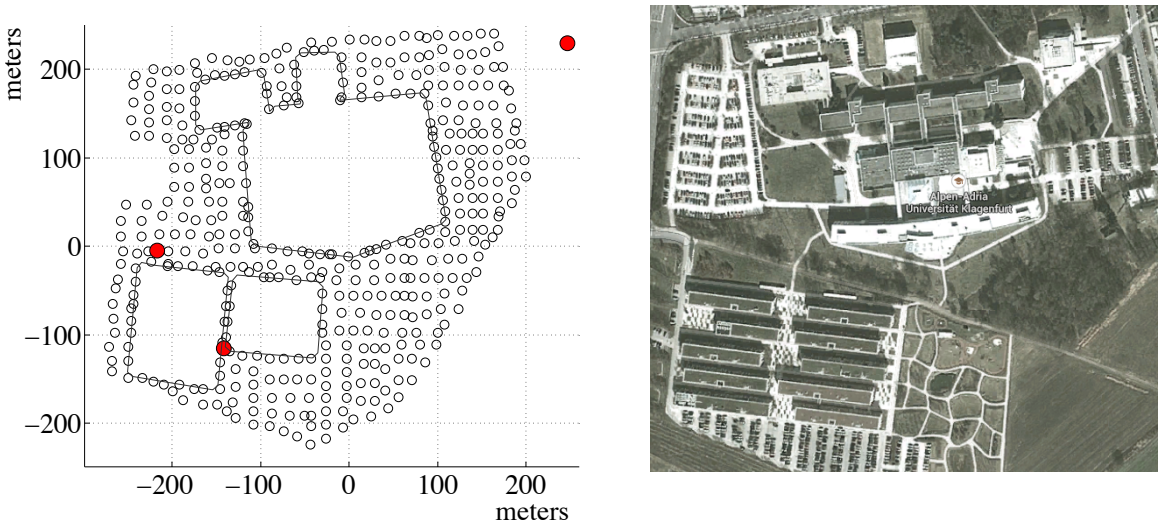


Figure 4.1: The *real-life* scenario "L_442p_3d_22b_6v". The right part of the figure is the satellite picture of the University of Klagenfurt taken from Google Earth™.

4.1.2 Tuning

Careful selection of the heuristic coefficients as well as the order and parameters of neighborhood operators might significantly improve performance of a method. This section is dedicated to tuning of the methods proposed for the CMP.

Selecting coefficients for the queue-based insertion heuristic (QI) The evaluation function (3.2.2) of this insertion heuristic is based on two semantically different measurements: distance and time. The first measurement is introduced to minimize the total travel distance. The second measurement is required to minimize the time delays between the visits. In order to achieve high performance, it is important to find a balance between the two parameters by setting the optimal values to their weight coefficients.

Values of the coefficients were selected as follows. For every coefficient a deterministic set of possible values is defined. This set is called domain of a coefficient. Then

the QI solves every training instance with all possible combinations of coefficients' values. The *real-life* scenarios with less than 100 points were used as a training set. Every combination is evaluated according to the costs of the obtained solutions.

The coefficients were set to values from the following domains. The domain of the coefficient α_1 was a set of the following values {0, 0.1, 0.3, 0.5, 0.7, 0.9, 1}. Value of coefficient α_2 can be computed as $\alpha_2 = 1 - \alpha_1$. The larger α_1 value, the more important distance to the point is and the less influence arrival time has. The last coefficient, *scale*, is responsible for balancing order of magnitude of the mentioned parameters. The coefficient scale was set to the following values {1, 10, 100}.

We evaluate the performance of each combination based on three measurements: number of times a combination returned the best solution, average and maximal percentage deviations from the best solution. The achieved results are reported in Table 4.1.

α_1	*scale*	Number of best solutions	Deviation from best solution, %	
			Average	Maximal
0	1	1	3.177	13.240
0	10	1	3.177	13.240
0	100	1	3.177	13.240
0.1	1	1	2.978	13.240
0.1	10	4	2.833	12.730
0.1	100	2	2.164	8.717
0.3	1	2	3.134	15.384
0.3	10	2	2.295	14.122
0.3	100	1	1.928	9.348
0.5	1	2	3.107	12.730
0.5	10	3	1.905	8.717
0.5	100	1	2.033	12.265
0.7	1	4	2.584	12.730
0.7	**10**	**5**	**1.461**	**7.218**
0.7	100	1	2.135	12.265
0.9	1	**5**	1.467	8.717
0.9	10	0	2.285	12.265
0.9	100	1	2.221	12.959
1	1	0	1.810	10.307
1	10	0	1.810	10.307
1	100	0	1.810	10.307

Table 4.1: Performance of different values of the coefficients of the QI heuristic

Performance of different combinations differs by fractions of percent on smaller instances. However, solution quality on a few larger instances varied significantly. It is indicated by values of the maximal deviation from the best solution. Therefore,

the maximal deviation was the main criterion of selection. The best combination of values was $\alpha_1 = 0.7$, $\alpha_2 = 0.3$ and $scale = 10$. Based on the selected values of the α coefficients, we can make conclusions about the most important impact factors for selecting a point-vehicle pair. Since the selected value of α_1 is greater than the values of other α-coefficients, the distance between a vehicle and a point is more important than the time-based parameters, i.e., arrival time and last visit time.

Selecting neighborhood structure for the variable neighborhood search Each neighborhood operator in a metaheuristic influences the solution differently. As a consequence, their correct order is significant for achieving improvements. The process of selecting the order of the operators applied within the proposed VNS metaheuristicis is discussed below.

The study was conducted on a training set consisting of the three randomly selected *real-life* scenarios: "L_46p_3d_3b_6v", "L_69p_3d_9b_6v", and "L_86p_3d_11b_6v". The initial solutions were constructed by the modified Clarke and Wright heuristic. Then the VNS solved each training instance with 10^4 iterations and every possible sequence of operators. In this thesis, the sequence of operators includes the *move* (m) and *cross-exchange* (c) operators 3 and 6 times, respectively, as in [41]. Since frequent application of the *insert* (i) operator minimizes the non-used energy per route, it is exploited in the sequence only 3 times. Otherwise, the number of feasible moves achievable by the other two operators can converge to zero. To minimize the influence of randomness produced by the operators, each instance was solved 10 times and the achieved results were averaged over all runs.

Each operator was evaluated individually and in combination with the others. The individual performance of every operator was measured as the number of times when the operator led to an improved solution. Each sequence of the operators was evaluated by their average percentage improvement per iteration and the total percentage improvement of one run.

Results achieved by the four best sequences are reported in Table 4.2. The sequence m-c-i produced the best average improvement per iteration and the largest final gain. The reason is that the *move* and *cross-exchange* operators precede the *insert* operator. They minimize the length of the routes that leads to more feasible moves for this operator. The *insert* operator decreases the optimization function value the most, as it increases the number of visits.

k	m-c-i	c-m-i	i-m-c	c-i-m
1	m 4	c 3	i 50	c 3
2	m 5	c 1	i 5	c 4
3	m 3	c 2	i 4	c 1
4	c 3	c 2	m 3	c 3
5	c 4	c 2	m 2	c 2
6	c 3	c 2	m 0	c 4
7	c 3	m 3	c 1	i 52
8	c 2	m 4	c 1	i 8
9	c 2	m 4	c 0	i 2
10	i 69	i 56	c 2	m 4
11	i 9	i 5	c 1	m 2
12	i 2	i 3	c 2	m 3
Aver. improvement	0.00121	0.00111	0.00114	0.000999
Total improvement	12.11268	11.13094	11.41604	9.989225

Table 4.2: Performance of the four best sequences of the neighborhood operators

4.1.3 Comparison with Optimum

When a heuristic is proposed, its performance should be assessed with respect to the state-of-the-art approaches [11]. In case if no method exists for the studied problem, one should provide an estimation of how close to optimum the heuristic's solutions are.

This study evaluates the solution costs provided by the QI, CW heuristics and their combinations with the VNS towards the optimum. The evaluation was conducted on the set of *optimum* instances. To find optimal solutions, we model the CMP in MiniZinc (G12 MiniZinc to FlatZinc converter version 1.6.0) as described in Section 3.5 and solve the model with the Gecode solver (version 4.2.1).

Figure 4.2 depicts the achieved performance as the deviation from optimum. The solutions computed by the QI and CW are at most 26.897 % and 41.413 % and on average 11.67 % and 25.337 % far from optimum, respectively. After 10^4 iterations the VNS significantly decreased the maximal deviations till 9.499 % for the QI and 16.802 % for the CW. The QI heuristic outperformed the CW on all scenarios except one and reached optimum on one instance.

The short computational time of the heuristics demonstrates the advantage of approximate algorithms. For instance, finding the optimal solutions took 4832.72 s on average, whereas the VNS required at most 1 ms for 10^4 iterations for every instance.

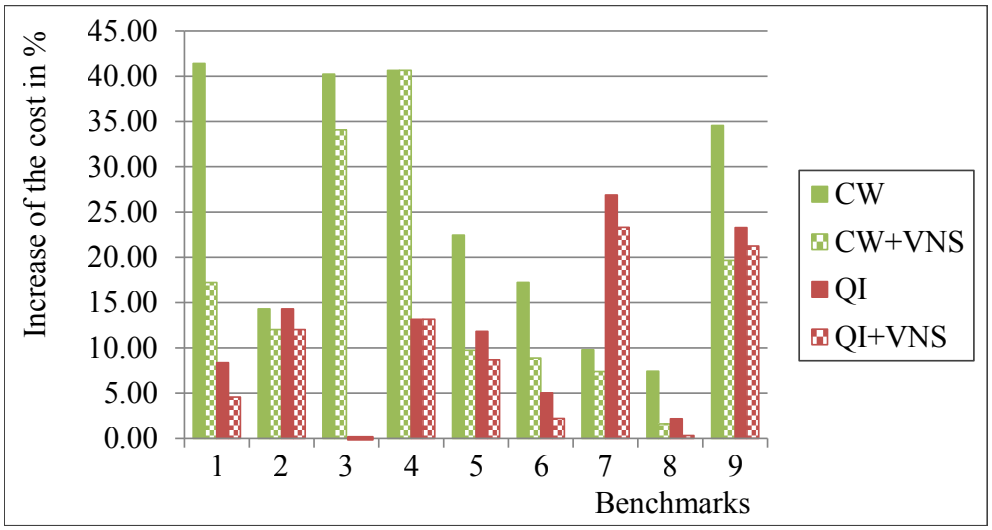


Figure 4.2: Deviation from optimum of the QI and CW heuristics and their combinations with VNS

4.1.4 Evaluation on Real-Life Instances

The practical CMP scenarios are significantly larger than the instances, where optimum can be computed. Therefore, it is important to estimate the performance of the proposed heuristics on realistic examples. This section reports the results achieved by the suggested construction heuristics QI and CW on a set of the *real-life* instances.

The evaluation criterion was the objective function of the CMP (2.1). We selected the best found solution for every instance and compare the heuristic to their cost.

The results obtained by both heuristics are shown in Table 4.3. Their performance differs depending on the instance size. The QI outperforms the CW heuristic on most of the scenarios with less than 90 points. It showed worse results only on 6 scenarios with over 85 points and the least number of vehicles. On the contrary, the CW obtained better solutions than the QI on larger instances. Its performance was worse only on two of such scenarios by around 6 % and 13.5 %.

Both heuristics require a very short computational time. For all instances, the QI and CW required only 0.355 s and 1.323 s, respectively.

Summarizing, both QI and CW heuristics are good candidates for real-life applications due to their low computational effort and ease of deployment. The preference should be given to one of them depending on the scenario size: the QI is more effective on the smaller instances, the CW provides better results on the larger benchmarks.

(a) Smaller instances with less than 90 points

	Number of obtained best solutions	Deviation from the best solution in %	
		average	maximal
QI	20	4.87	37.16
CW	6	7.29	18.15

(b) Larger instances

	Number of obtained best solutions	Deviation from the best solution in %	
		average	maximal
QI	2	6.89	16.90
CW	8	1.95	13.47

Table 4.3: The performance of the QI and CW heuristics on the *real-life* scenarios

4.1.5 Summary

This section presents the evaluation results of the approximate algorithms developed for solving the continuous monitoring problem. Among the two construction approaches, the QI heuristic achieved better results for the smaller real-life instances and when compared with optimum. The second heuristic CW was more efficient on larger real-life instances. We suggest to apply one of these heuristics depending on instance size.

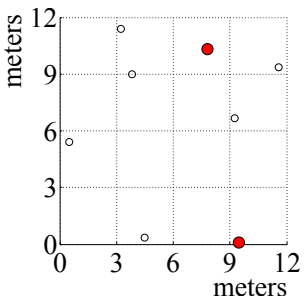
The VNS metaheuristic significantly improved the initial solutions provided by the QI and CW. The costs decreased by up to 24 % for the *optimum* instances and 12 % for the *real-life* instances.

4.2 CMP with Inter-Depot Routes

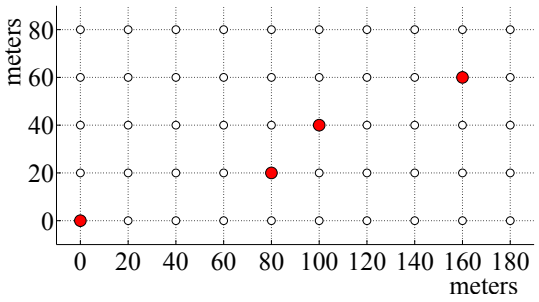
4.2.1 Test Instances

To make a thorough evaluation of our method, five sets of instances[2] were generated: 10 small *optimum* instances, 12 *patrolling*, 48 *life*, 60 *random*, and 24 *clustered* scenarios. The first set is used to compare the proposed heuristic IDIH-Reserve with optimum, whereas the other sets are larger in size and are used to estimate performance on realistic examples. Fig. 4.3 depicts examples of the instances.

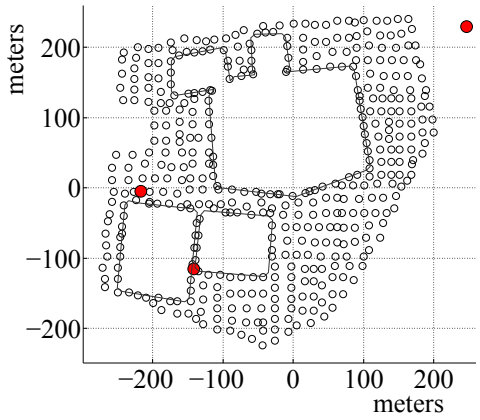
[2]All sets are available online at http://uav.lakeside-labs.com/publications/test-data/planning/



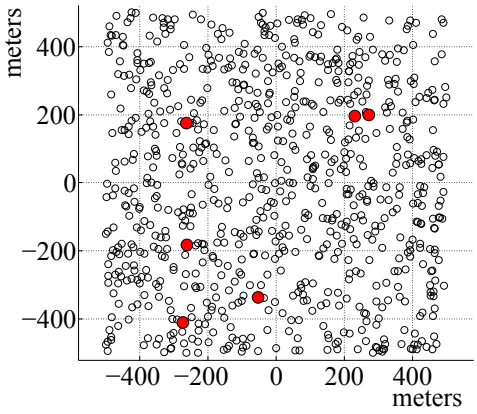
Set 1: *optimum*



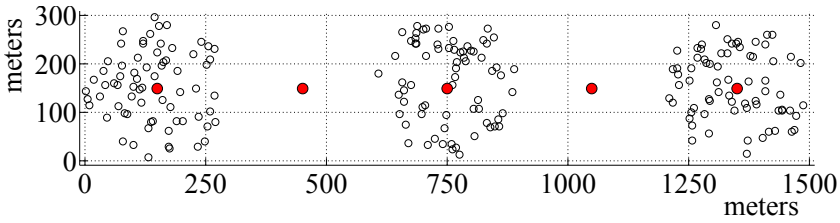
Set 2: *patrolling*



Set 3: *real-life*



Set 4: *random*



Set 5: *clustered*

Figure 4.3: Examples of instances from the five test sets. Red circles denote stations, white circles represent picture points.

Optimum instances are significantly smaller than real-life scenarios. Therefore, it is possible to calculate optima in a feasible amount of time. This set consists of 10 instances with 6 points and 2 stations. Coordinates for the picture points and the base stations were selected randomly in the interval $[0, 12]$. All points are split in two clusters so that the first and the second clusters have priorities 1 and 2, respectively.

Each scenario has 5 batteries and 2 homogeneous vehicles. The batteries are randomly allocated at the stations. Their capacity is sufficient for several overview images. The vehicles have average speed, battery change time, and service time equal to 1. Their initial locations and remaining energy capacities as well as the delay lvt_p of each points are selected randomly.

Since an optimum cannot be found for common real-life scenarios, the IDIH-Reserve solutions are compared with the optimal solutions for the patrolling task. It is a related problem that relaxes some constraints, thus, providing a lower-bound for the CMPIDP optimum. The vehicles are homogeneous with *unlimited* battery capacity. The goal of patrolling is to compute a set of routes for a fleet of vehicles to repeatedly visit a set of points with a minimal average delay.

The **patrolling** instances are generated so that the solutions provided by the TSP-based strategy by Almeida et al. [3] are guaranteed to be optimal. This strategy constructs a TSP route through all the points and places the robots along it with equal distances between each other. It guarantees equal and minimal delays between visits.

In order to generate the aforementioned *patrolling* instances, the following steps were performed:

1. *Placing the points* The points are placed in a grid with 20 m step size. The total number of points is within 45–375. Each point has the same priority, and the parameter lvt_p is equal to zero.

2. *Placing the stations* The first station is placed at the left-most bottom point with coordinates $(0, 0)$. The remaining four stations are placed along the optimal TSP route with equal distances between them. Fig. 4.4 shows an example of a TSP-route and the corresponding stations.

3. *Generating vehicles and batteries* One vehicle is initially located in each station. The vehicles are homogeneous and move with the average speed 1 m/s. Since patrolling ignores battery capacity limitation, this constraint is relaxed as follows. The battery capacity is equal to the travel time between the stations. The service time and the time to change the battery are equal and set to 0 s.

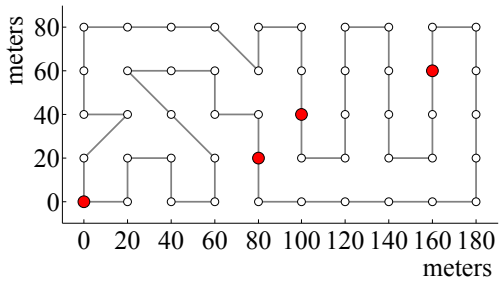


Figure 4.4: An example of a TSP route for generating the *patrolling* instances.

4. *Setting the planning horizon* The IDIH-Reserve was developed for the problem with limited number of energy resources. The patrolling strategy, on the contrary, does not set any upper bound on the planning horizon. For a fair comparison, the mission time is set to several values: the travel time of the TSP route multiplied by 2, 4, 6 or 8.

The third set (**real-life scenarios**) consists of several scenarios that we observed in practice, e.g., an area around the University of Klagenfurt or a construction site near Vienna. More details on the picture point generation for this set can be found in Section 4.1.1.

The picture points of the *random* scenarios are placed randomly with coordinates in the following intervals: $[-300, 300]$ for 200–300 points, $[-400, 400]$ for 301–600 points, $[-500, 500]$ for 601–800 points. The *random* scenarios have 200–800 points.

The remaining parameters of the *real-life* and *random* instances are generated as follows. Three or six base stations are randomly allocated with coordinates in the mentioned intervals. Each scenario has 4, 7 or 8 vehicles of 2, 3 or 4 types, respectively. The number of batteries is chosen so that the area can be covered 3 or 6 times. The batteries are either assigned to the stations randomly or distributed among them evenly. Depending on the type, the maximal flight time with one battery is either 1200 s or 2400 s. For simplicity all vehicles have the same average speed of 2.5 m/s. Their initial location and remaining energy capacity, as well as the delay lvt_p of the points are generated randomly.

Clustered scenarios were generated to evaluate the algorithm performance in extreme cases when the points are grouped in clusters and a vehicle has to visit an intermediate station to travel between the clusters.

We have used two types of cluster allocation: triangular and sequential. Fig. 4.5 illustrates both types. As shown in the figure, every triangular scenario has 6 base stations, whereas every sequential scenario has 5 stations. Every cluster has 70, 130

or 200 points placed randomly within the boundary of the cluster depicted as a dotted line. Batteries sufficient to cover the area at least 3 or 6 times are assigned to the stations randomly. Each scenario contains 2 or 4 vehicles.

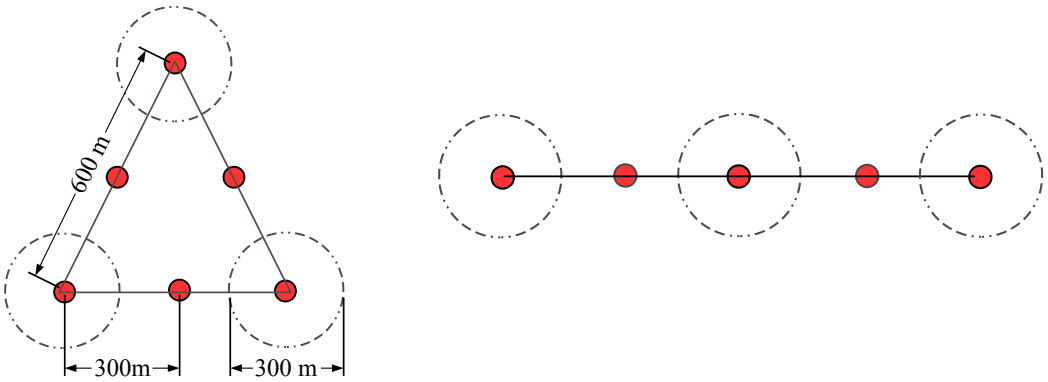


Figure 4.5: Allocation of clusters in the *clustered* scenarios: triangular and sequential.

All large instances are named according to the following rule. The first letter stands for the instance type: "P" for *patrolling*, "L" for *real-life*, "R" for *random* and "C" for *clustered* scenarios. The next three numbers define the number of points, stations and vehicles. The last number stands for the minimal number of times that the whole area can be covered with the given batteries. For the *real-life* and *random* scenarios an additional letter is introduced. It defines whether the batteries are distributed randomly or equally among the stations. For example, a file named "L_251p_3s_4dr_3ov_E" is a *real-life* scenario with 251 points, 3 stations, 4 vehicles, where the area can be covered at least 3 times. The stations have equal number of batteries of each type.

Names of the *optimum* instances are simplified, as all the parameters used for naming the large instances have the same values. They are generated as a combination of the letter "O" that stands for optimum and the sequential number of an instance. For example, the first *optimum* instance is named as "O_01".

4.2.2 Tuning

Selecting coefficients for the construction heuristics Typically, parameter tuning results in noticeable improvements. In the following, we describe the process of selecting the best-performing parameters for the IDIH and IDIH-Reserve heuristics for the CMPID problem: coefficients α_1, α_2 and $\alpha_3 = 1 - \alpha_1 - \alpha_2$. For this process we focused on the *life*, *random* and *clustered* instances. In order to avoid over-fitting, every third instance was excluded. Every instance was solved with its initial mission

time mt as well as $mt \cdot 1.5$ and $mt \cdot 2.5$. We evaluate all possible combinations of the coefficients from the interval $[0, 1]$ with the step 0.1.

The best performing combination of values was selected in two stages. The first stage determines the most efficient combination for every instance set and mission time value. In the second stage we make the final selection as described later.

The first stage starts by eliminating the combinations that did not consistently perform among the best. The performance is based on the average deviation from the best solution found for each instance. The next step selects three best combinations for every instance set and mission time value. The best combinations must have the least maximal deviation among all instances and one of the least average deviations. Results of this selection are reported in Tables 4.5 and 4.6 for the IDIH and IDIH-Reserve, respectively.

Set of instances	Mission time multiplier	α_1	α_2	α_3	Average deviation in %
real-life	1	0.5	0.1	0.4	19.207
		0.2	0.1	0.7	21.468
		0.5	0.3	0.2	22.363
	1.5	0.5	0.1	0.4	20.645
		0.5	0.4	0.1	23.455
		0.7	0.3	0	23.687
	2.5	0.9	0.1	0	16.664
		0.7	0.3	0	18.770
		0.5	0.1	0.4	20.607
random	1	0.4	0.1	0.5	18.420
		0.6	0.1	0.3	20.425
	1.5	0.8	0.2	0	28.898
		0.4	0.1	0.5	31.932
		0.2	0.1	0.7	32.988
	2.5	0.8	0.2	0	19.546
		0.4	0.1	0.5	23.507
		0.7	0.3	0	25.005
clustered	1	0.4	0.6	0	23.792
		0.3	0.6	0.1	24.103
		0	0.3	0.7	24.795
	1.5	0.4	0.6	0	24.701
		0	0.6	0.4	25.658
		0.2	0.8	0	26.480
	2.5	0.2	0.8	0	16.001
		0.4	0.6	0	17.536
		0.7	0.3	0	20.792

Table 4.5: Preliminary selection of α-coefficients for the IDIH heuristic

Set of instances	Mission time multiplier	α_1	α_2	α_3	Deviation in %	
					average	maximal
real-life	1	0.3	0.6	0.1	2.281	14.862
		0.4	0.5	0.1	2.920	24.984
		0.2	0.7	0.1	3.195	26.887
	1.5	0.3	0.6	0.1	2.733	7.409
		0.4	0.5	0.1	2.030	7.855
		0.5	0.4	0.1	3.395	8.510
	2.5	0.6	0.3	0.1	1.755	5.369
		0.7	0.2	0.1	1.912	5.428
		0.3	0.6	0.1	2.535	6.389
random	1	0.2	0.7	0.1	3.545	7.681
		0.5	0.4	0.1	2.982	10.606
		0.4	0.5	0.1	2.720	13.034
	1.5	0.4	0.5	0.1	2.418	8.141
		0.3	0.6	0.1	3.077	8.430
		0.2	0.7	0.1	3.256	8.542
	2.5	0.5	0.4	0.1	1.229	4.147
		0.4	0.5	0.1	1.659	5.516
		0.7	0.2	0.1	1.977	6.322
clustered	1	0.4	0.2	0.4	3.550	11.870
		0.5	0.2	0.3	3.474	11.903
		0.6	0.3	0.1	5.832	13.687
	1.5	0.5	0.2	0.3	4.574	8.131
		0.1	0.8	0.1	3.949	8.807
		0.5	0.3	0.2	4.366	10.662
	2.5	0.5	0.4	0.1	3.581	8.598
		0.6	0.3	0.1	3.360	9.801
		0.2	0.7	0.1	2.630	11.932

Table 4.6: Preliminary selection of α-coefficients for the IDIH-Reserve heuristic

During the second stage we analyzed the results obtained in the previous section and could derive performance patterns. The coefficients of the IDIH heuristic showed quite high dependency on distribution of points, whereas performance of the IDIH-Reserve coefficients differed only between clustered and non-clustered instances. Table 4.7 presents the coefficients value that we suggest based on the analysis.

(a) IDIH

Set of instances	α_1	α_2	α_3
real-life	0.5	0.1	0.4
random	0.4	0.1	0.5
clustered	0.4	0.6	0

(b) IDIH-Reserve

Set of instances	α_1	α_2	α_3
real-life	0.4	0.5	0.1
random	0.4	0.5	0.1
clustered	0.5	0.2	0.3

Table 4.7: The best coefficients values for the QI and CW heuristics

Selecting neighborhood structure A neighborhood structure specifies the neighborhood of the current solution where the search process looks for a better solution. It can be given either implicitly by listing all neighboring solutions or explicitly by a sequence of modification operators. Metaheuristics use the second way of exploring the search space.

A good neighborhood structure increases efficiency of a metaheuristic [37]. The order of the operators influences the shape of a neighborhood, whereas parameters of the operators define its size. In the following we describe the procedure to define both the order and the parameters of the used VNS operators (see Section 3.3.3).

The study is conducted in two steps. The first step selects the best value of the parameter η for every operator: *move*, *cross-exchange*, *replace* and *insert*. This parameter stands for the maximal length of a sequence(s) for the first two operators and the maximal number of points for the other two moves. We also use this parameter as an indicator of how often each operator is used in the neighborhood sequence. The second step derives the best order of the operators.

The mentioned steps have the following setup. Initial solutions are constructed by the IDIH-Reserve heuristic. Since it is a deterministic algorithm, solution quality is not biased by the construction heuristic. Then the VNS performs the improvement stage with 10^4 iterations. The influence of the random components of the VNS operators is minimized by running the metaheuristic ten times with every used setting. The obtained results are averaged over all runs. The studies are conducted on the 10 instance from the *real-life* set. For every number of points, we select the largest instance, i.e., with 6 stations and 7 vehicles that can cover the area at least 6 times.

During the first step, the best value of the parameter η is selected from the set $\{1, 2, 3\}$. Performance of different η values is measured as the average percentage improvement from the initial solution cost. The results are presented in Figure 4.6 for all operators. In this case parameter η does not significantly effect the performance. Therefore, it is set to value 1 for all the operators to encourage frequent neighborhood change.

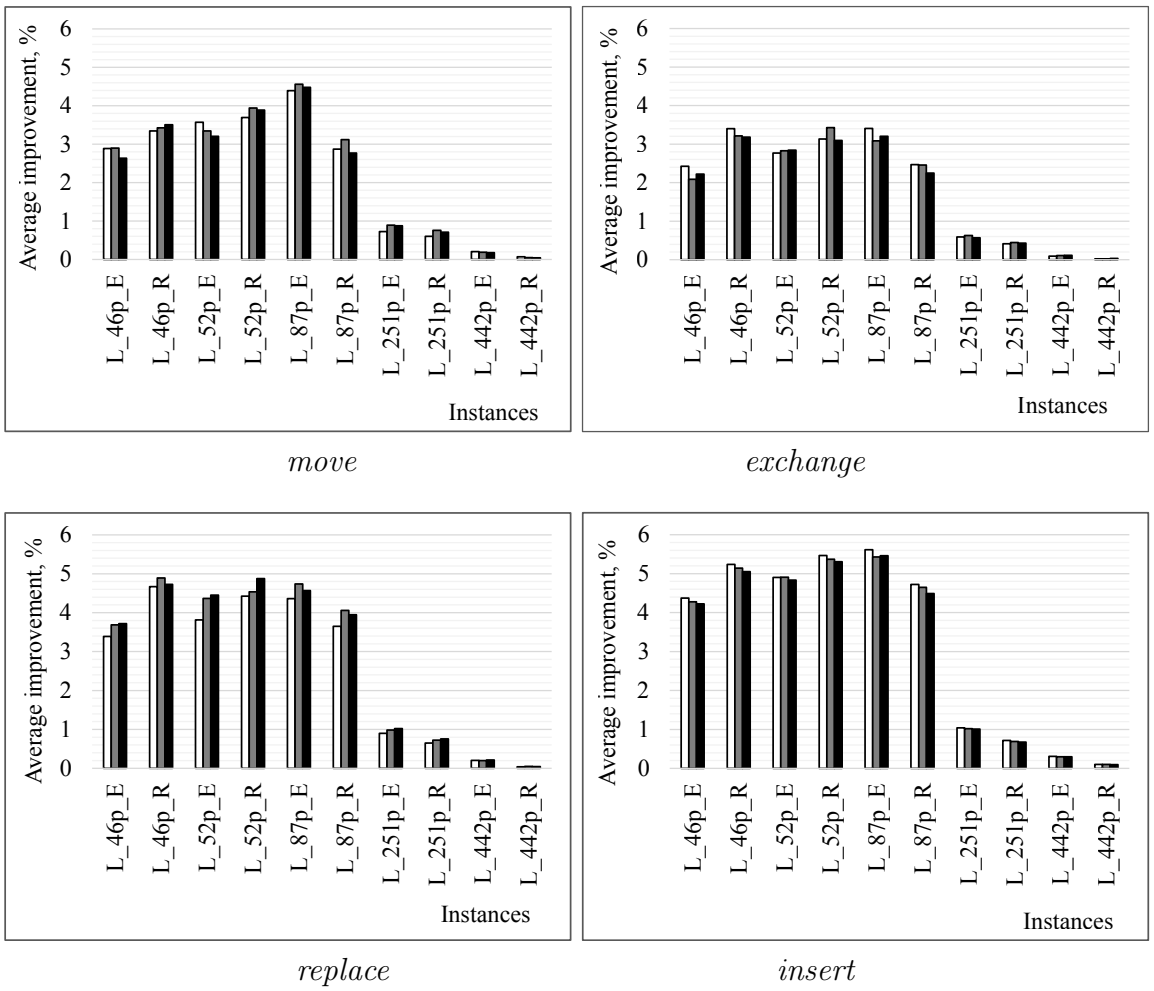


Figure 4.6: Average total improvement from a single operator with parameter η equal 1 (white), 2 (gray) and 3 (black).

The goal of the next step was to select the best performing sequence of the operators. Performance of each sequence was measured as the solution cost on each instance. We also evaluate the contribution of each individual operator by the number of its uses that lead to an improvement.

The improvement achieved by different sequences does not differ significantly after the 10^4 iterations. Their deviation from the best average improvement is less than

1 %. This means that the improvement achieved by the VNS is not influence by the order of its operators in a general case of the CMPID. For further evaluations the order *insert-exchange-move-replace* is selected, as for the most instances, the number of improving moves decreases with increasing operator index. This is the expected behavior in a sequence of neighborhood operators [41].

4.2.3 Comparison with Optimum

In order to solve our real-life instances in acceptable time we have to apply heuristics which may result in non-optimal solutions. Therefore, we evaluate how far from optimum the proposed heuristics can get.

The IDIH and IDIH-Reserve heuristics are compared with optimum on *optimum* instances. The heuristics use the coefficient values chosen in the previous section. Their performance is evaluated as the deviation from the cost of optimal solutions that is reported in Table 4.9.

Instance	Optimal cost	IDIH		IDIH-Reserve	
		cost	deviation	cost	deviation
O_01	92121	98649	7.09	98649	7.09
O_02	97044	99036	2.05	99036	2.05
O_03	7984	9876	23.70	9210	15.36
O_04	9656	10804	11.89	9976	3.31
O_05	8158	8854	8.53	8528	4.54
O_06	7832	7912	1.02	8390	7.12
O_07	9132	9964	9.11	9964	9.11
O_08	6465	8985	38.98	7257	12.25
O_09	8333	8565	2.78	8931	7.18
O_10	7865	7943	0.99	7943	0.99

Table 4.9: Deviation from optimum of the solutions obtained by the IDIH and IDIH-Reserve heuristics

The IDIH-Reserve heuristic outperformed the IDIH on 8 out of 10 instances. The difference in performance is the most significant on the two hardest instances "O_03" and "O_08". In these instances, points are distributed in such a way that most of the points can be reached by only a single UAV. In this case it is extremely important how heuristics handle the batteries and select the points. More discussion on that is provided in Section 4.2.4.

As expected, computation time differs significantly between the solver and the two heuristics. Heuristics require milliseconds, whereas solver spends hours even for the smaller scenarios. This is a good illustration of the problem complexity. It shows

how important it is to find a balance between the solution quality and computational efforts.

4.2.4 Evaluation on Real-Life Instances

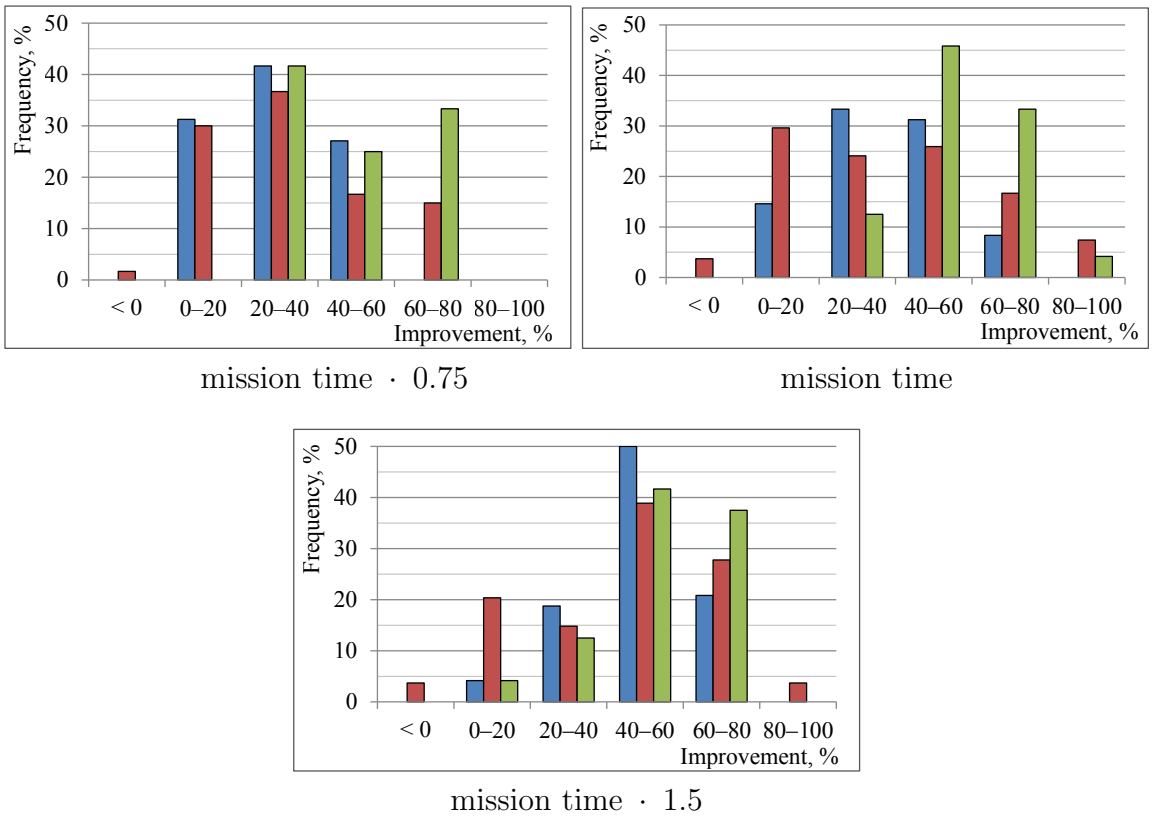
In this section we present the evaluation of the proposed heuristics on large instances. First, the heuristics are compared between each other. Next, we evaluate their solution quality on the problem without the inter-depot routes (CMP) and the patrolling task problem. The section concludes with the scalability test for the VNS.

Comparison of the construction heuristics In this study we compare the performance of the IDIH and IDIH-Reserve heuristics. Both of them iteratively construct a solution by adding one picture point at a time. A point and a place where it is inserted are selected based on an evaluation function. The differences between the two approaches are in the parameters of this function and a battery reservation procedure implemented by the IDIH-Reserve.

The study had the following settings. It was conducted on the *real-life*, *random* and *clustered* instances. Their mission time had three possible values: its original value mt as described in Section 4.2.1 as well as $mt \cdot 0.75$ and $mt \cdot 1.5$. The heuristics used the coefficients values suggested in Section 4.2.2. Their performance was assessed by the CMPID goal function.

Figure 4.7 shows the achieved results. They are grouped by the mission time value. The Y-axis depicts percentage of instances that were solved by the IDIH-Reserve with the corresponding improvement over the IDIH (X-axis). The IDIH-Reserve outperformed the IDIH on almost all instances with up to 87.6 % improvement. The improvement is significant regardless of the instance size, the distribution of the picture points or the mission time.

There are two major reasons for such significant improvement. First, due to a reservation policy introduced in the IDIH-Reserve, all vehicles have a spare battery at every step. In contrast, the IDIH can extend vehicle's route relying on the last battery at a station that can be taken by another drone at the next step. In this case, the first vehicle has to finish its mission as soon as its battery is discharged. This can happen at any time during the mission. This leads to a non-optimal use of batteries. The second reason is that the evaluation function includes relative time-based parameters that do not increase with the mission time unlike absolute parameters. Absolute time-based arguments (arrival time and last visit time) influence selection of the insertion point more, as mission proceeds. With long mission duration, they can completely overrule the distance parameter which makes routes less efficient. Fig. 4.8 illustrates the effect



mission time · 0.75 mission time

mission time · 1.5

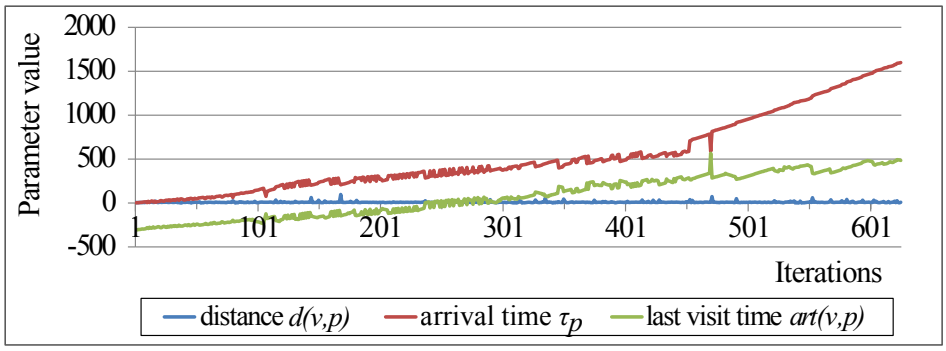
Figure 4.7: Improvement of the IDIH-Reserve heuristic over the IDIH on *random* (blue), *real-life* (red) and *clustered* (green) instances.

of relative measurements on the instance "L_251p_3st_4dr_3ov_E". This figure shows how values of the parameters change in both the IDIH and IDIH-Reserve approaches. The IDIH time-based parameters are constantly increasing, whereas the IDIH-Reserve parameters stay within certain boundaries. Therefore, we can conclude that the IDIH is highly dependent on the instance size unlike the IDIH-Reserve.

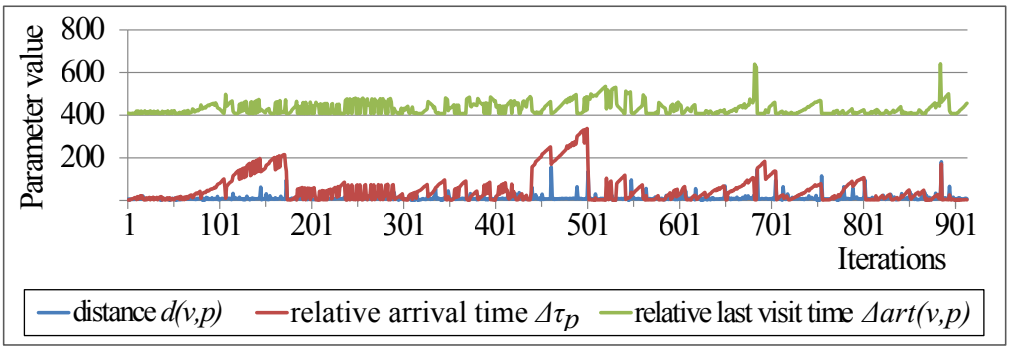
Comparison with the CMP approaches This study estimates the benefit of introducing inter-depot routes to the continuous monitoring problem. It compares the IDIH+VNS approach with the CMP algorithms, i.e., QI+VNS and CW+VNS.

We conducted this evaluation as follows. The test set consisted of the *real-life* and *random* instances. The CMP methods were used with the parameters suggested in Section 4.1.2. We ran the VNS for 10^4 iterations 10 times for each scenario, and the results are then averaged over all runs. The performance of the heuristics are compared by both solution cost, i.e. the CMP goal function (2.1), and computational time.

Figures 4.9 and 4.10 summarize the achieved results: the cost deviation from the best found solution and the computational time, respectively. The figures cover only



IDIH



IDIH-Reserve

Figure 4.8: Value of the parameters of the evaluation functions on the instance "L_251p_3st_4dr_3ov_E".

a selection of the instances, in particular scenarios with random battery allocation. The results on other instances are similar and, thus, this selection can be used to reason about the heuristics' performance.

According to Figure 4.9 depicting the cost deviation from the best solution, the CMPID heuristic significantly outperforms the CMP approaches. This applies to all but two instances. There are two reasons for that. First of all, the VNS for the CMPID has a different neighborhood structure that provides more improvements. The second reason is the problem extension with the inter-depot routes. In the CMP approaches, vehicles are forced to utilize only batteries of their home stations. As a consequence, the picture points are visited with different frequencies depending on the distribution of batteries and vehicles. On the contrary, the CMPID heuristic allows traveling between the stations and, consequently, uses the given energy resources more efficiently.

Figure 4.10 shows that the CMP methods require longer computational time than the IDIH+VNS approach. It is caused by the local search step included in the VNS for the CMP. According to our preliminary results, the improvements achieved by

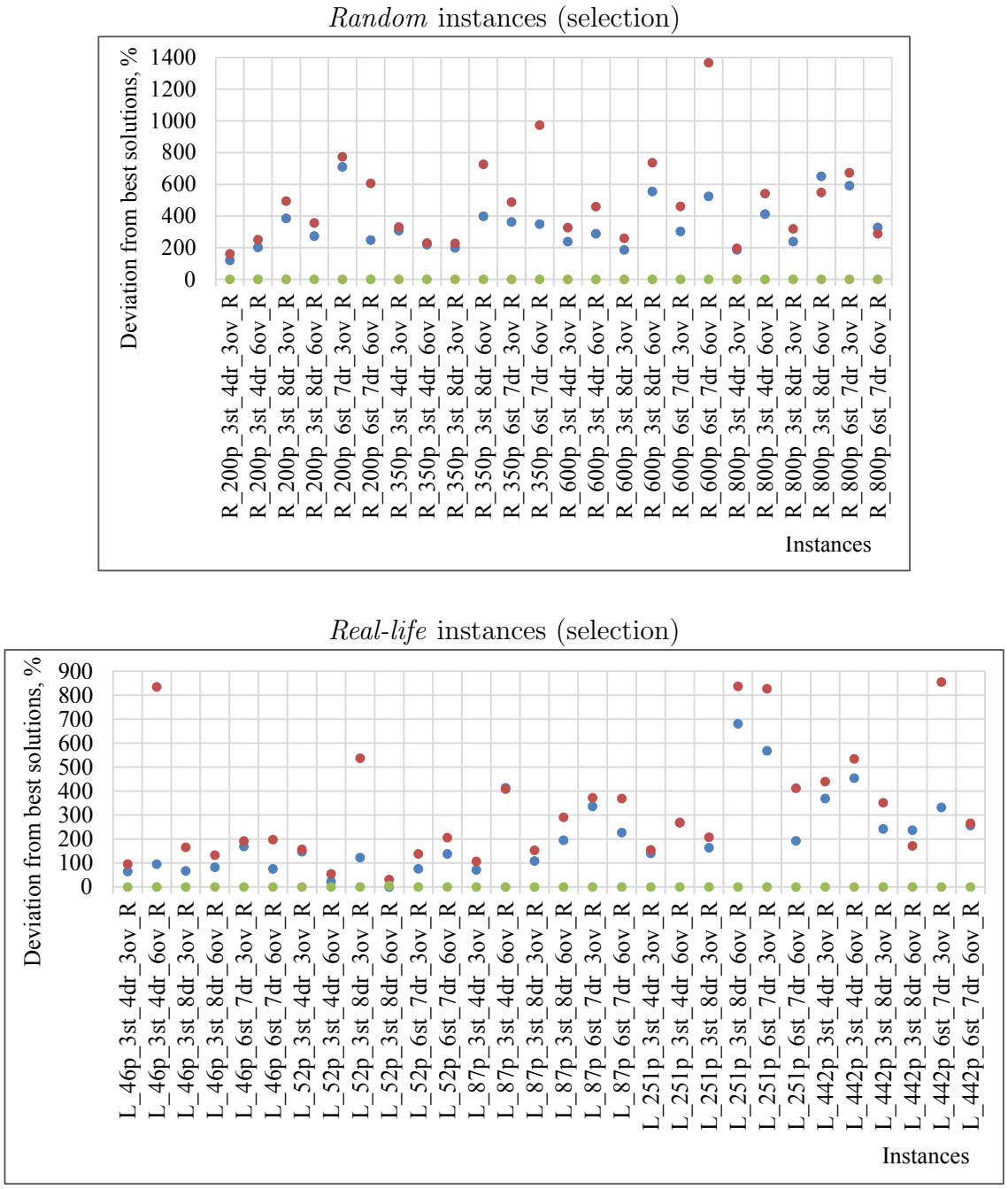


Figure 4.9: Comparison of the solution costs achieved by the IDIH+VNS (green) and the CMP approaches, i.e., QI+VNS (blue) and CW+VNS (red).

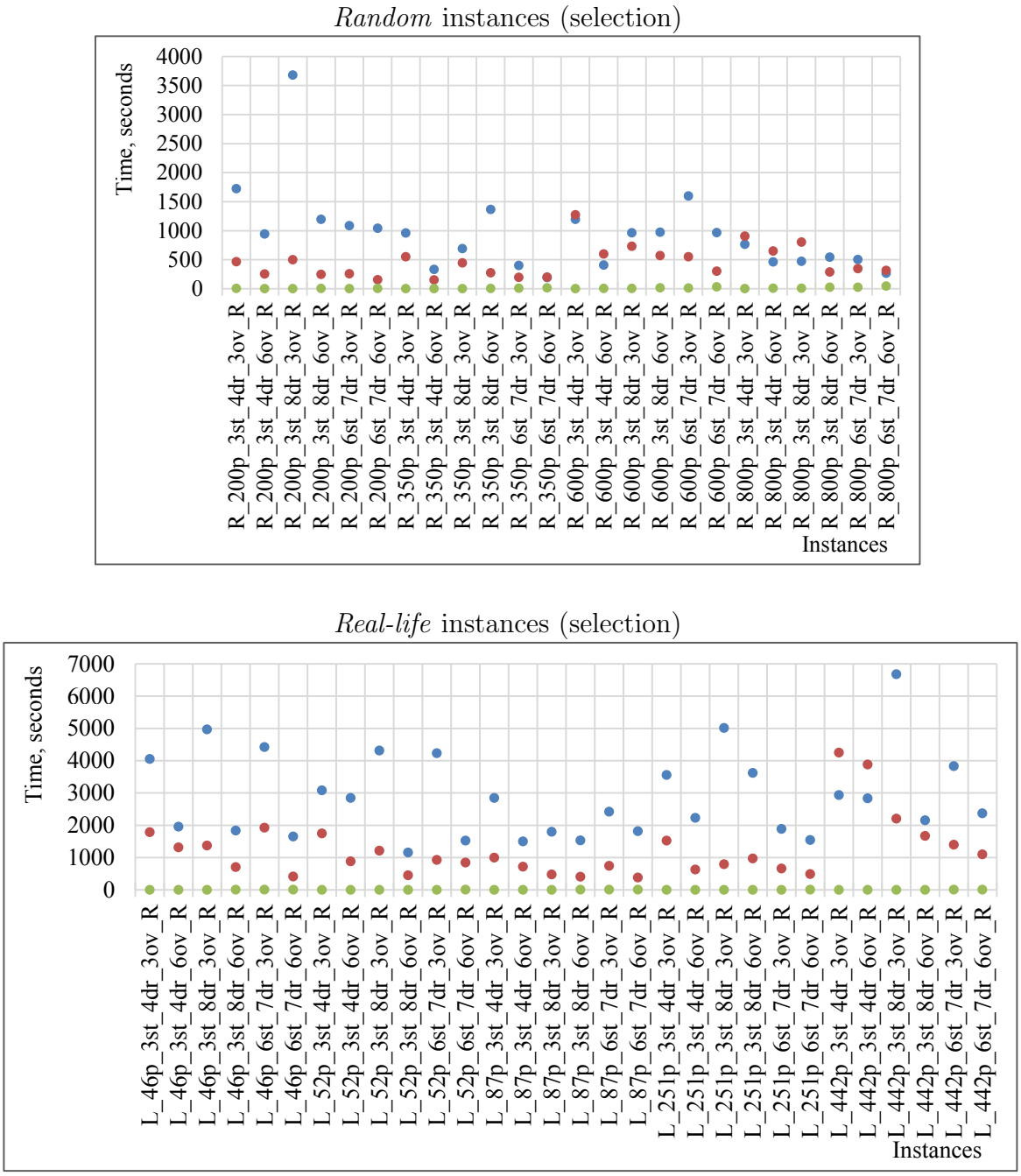


Figure 4.10: Comparison of the computational time required by the IDIH+VNS (green) and the CMP approaches, i.e., QI+VNS (blue) and CW+VNS (red).

the local search in the VNS for the CMPID are negligible comparing to the computational efforts. Therefore, this step is omitted for the CMPID version of VNS. If the CMP VNS is applied without local search, it requires similar run-time as the CMPID version.

Comparison with the patrolling strategy This study evaluates the proposed heuristics on the patrolling task problem that, similar to the CMPID, minimizes the average delay between visits of points and variance of delays. However, in contrast to the monitoring problem, patrolling assumes unlimited battery capacities.

For this evaluation, we use the TSP-based patrolling strategy by Almeida et al. [3] as a basis. It first creates a TSP route through all target points. Then robots are placed at some locations along the TSP route with equal distances between them. The TSP-based patrolling strategy and the tailored *patrolling* instances guarantee solutions with equal and minimal delays between consecutive visits. We employ these instances as stress tests to evaluate the solutions of our more general methods. Since the IDIH-Reserve heuristic outperformed the IDIH, we use only IDIH-Reserve for this comparison.

The performance of the IDIH-Reserve and the TSP-based strategy is compared regarding the average delay between the visits. The box plots in Fig. 4.11 show the results achieved by the IDIH-Reserve. The bold lines depict the delays of the TSP-based strategy. As expected due to the setup of the study, our heuristic deviates from the patrolling solution. In particular, this deviation increases only linearly with the number of visits. The average delay is only 10.72 % above the optimum on the largest scenario with 8 rounds corresponding to 4 hours mission time.

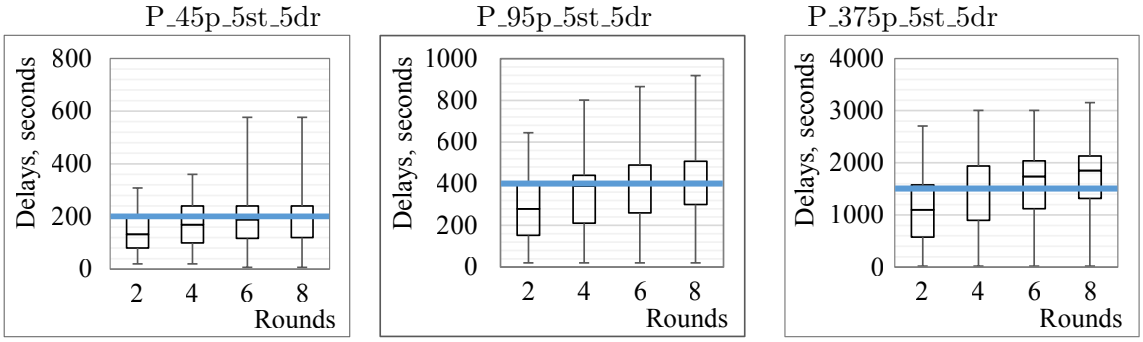


Figure 4.11: Average delays for the *patrolling* instances.

Scalability of the VNS The final evaluation was conducted to check the scalability of the proposed improvement step, i.e., how well it performs on different scenario sizes.

This study was conducted on the *real-life* and *random* scenarios as follows. First, the IDIH computes an initial solution and its value of the goal function for each scenario. Then the VNS is ran 10 times, for 10 minutes in each run. The obtained improvement, i.e., deviation of the cost from the initial value, is averaged over these 10 runs.

The results are reported in Fig. 4.12. The left and the right charts show the achieved cost improvement for the scenarios with equal and random resource allocation, respectively. The proposed VNS provides improvements to the initial solution regardless of scenario size. Furthermore, the computational time used to achieve these results was only 10 minutes. Thus, a team of first responders can use this metaheuristic at the initial stage of the rescue operation when the UAVs are getting ready to take off.

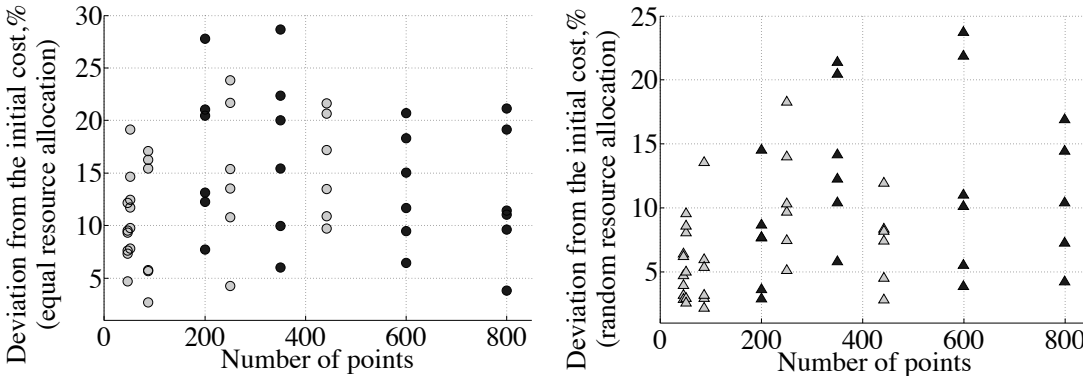


Figure 4.12: Improvements of the solution quality after 10 minutes. Gray markers: *real-life* scenarios; black markers: *random* scenarios.

4.2.5 Summary

This section assesses solution quality and other characteristics of the heuristics developed for the CMPID problem. When compared with optimum on smaller instances, the IDIH-Reserve deviates from optimum by only 6.9 % on average and at most 15.36 %. On larger instances including several real-life scenarios, the IDIH-Reserve outperforms the other construction heuristic IDIH. This is due to a battery reservation mechanism and another evaluation function.

The proposed metaheuristic VNS improves initial solutions for large random and real-life instances obtained by the IDIH on 3–30 % in only 10 minutes.

A combination of the VNS and IDIH outperforms the approaches for the CMP, QI+VNS and CW+VNS in 3 times in solution quality and in 481 times in computa-

tional time. This shows that introducing inter-depot routes increases efficiency, i.e., vehicles visit more target points at a more uniform frequency. In our application this leads to more recent information updates which are important for situations where a small missed change can be critical.

The next study compares the performance of the proposed heuristic with the optimal solutions of the patrolling problem. The used instances were generated by relaxing the essential energy constraints. Even though these instances are favorable for patrolling, the average delay of the IDIH-Reserve solutions deviates by only 11 % from the optimal patrolling strategy on the largest scenarios.

4.3 CMPID with Priorities

4.3.1 Test Instances

The CMPIDP is a variation of the CMPID problem with priority information and without the fixed mission time. Therefore, the CMPID instances from Section 4.2 can be used for evaluating the CMPIDP methods with minor changes. In this section we involve all benchmark sets except the *patrolling* scenarios. The following modifications were made to the instances: we omit the mission time limit and include additional information about priorities of the points.

Priorities are assigned as follows. For the *optimum* instances, the picture points are split in two clusters so that the first and the second clusters have priorities 1 and 2, respectively. The *real-life*, *random* and *clustered* instances include 3 priority levels. To assign them, the area is split into either 3 clusters (for *clustered* scenarios and scenarios with up to 200 points) or 6 clusters. All points in a cluster have the same priority from the set $\{1, 2, 3\}$. For the instances with 6 clusters each priority level has to be assigned to exactly two clusters.

4.3.2 Tuning

Typically, parameter tuning leads to noticeable improvements. In the following, we describe the process of selecting the best-performing coefficients for the IDIH-Reserve heuristic for the CMPIDP.

The study was conducted on three benchmark sets: the *real-life*, *random* and *clustered* instances. In order to avoid overfitting, every third instance was excluded.

Tuning of coefficients is typically performed by comparing the performance of a heuristic with all possible combinations of coefficient values from a fixed domain.

In this study, all coefficients (except *scale*) took values from the interval $[0, 1]$ with step 0.1. The coefficient *scale* was set to 100. The performance of a combination on an instance was measured by the cost deviation from the best obtained solution on this instance. The best combination has minimal average and minimal maximal deviations among all instances.

The obtained results showed that the instances require different coefficients depending on instance size and points distribution. Table 4.10 reports the suggested values for the subgroups of instances.

Set of instances	α_1	α_2	α_3	α_4	β
clustered	0.2	0.2	0.2	0.4	0.9
non-clustered ($>$ 100 points)	0.3	0.4	0.2	0.1	0.7
non-clustered ($<$ 100 points)	0.2	0.6	0.1	0.1	0.7

Table 4.10: Coefficients of the IDIH-Reserve for the CMPIDP

According to the selected values of the α-coefficients, the arrival time dominates the other parameters of the evaluation function on the *real-life* and *random* instances. On instances with more scattered picture points, the parameters have almost equal impact. This change of influence is a mechanism to deal with bottleneck situations like in our *clustered* scenarios.

4.3.3 Comparison with Optimum

This evaluation estimates performance of the IDIH-Reserve heuristic with respect to optimum. We use the *optimum* instances, where the optimal solutions were found by solving a MiniZinc model (see Section 3.5) with the Gecode solver. Solution quality is measured by the CMPIDP goal function.

The achieved solution costs are reported in Table 4.11. They show that the proposed heuristic is on average only 4.09 % and at most 13.66 % worse than the optimum. Moreover, the heuristic requires less than a second to solve each instance instead of minutes or hours required by the solver.

4.3.4 Evaluation on Real-Life Instances

In the presence of priorities, areas of higher importance should be visited more often. This study shows that the IDIH-Reserve provides such solutions with the update frequency proportional to the priority level.

The *real-life*, *random* and *clustered* instances were solved twice. In the first run, average delays for each of the three priorities were determined. The second run

Instance Nr.	Optimal cost	IDIH-Reserve	
		cost	deviation in %
1	30364	30884	1.71
2	23441	23961	2.22
3	17267	17983	4.15
4	19073	19497	2.22
5	18378	19156	4.23
6	23795	24239	1.87
7	25038	26074	4.14
8	19348	20426	5.57
9	23158	26322	13.66
10	20111	20335	1.11

Table 4.11: Comparison with optimum

ignored the priority information and, thus, all points were equally important. The performance was measured as the average delay between visits of points of the same priority.

Fig. 4.13 demonstrates the results of the study. Average visit frequencies of the first run are shown as percentages of the visit frequency of the second run. These charts also show the variations among the measurements.

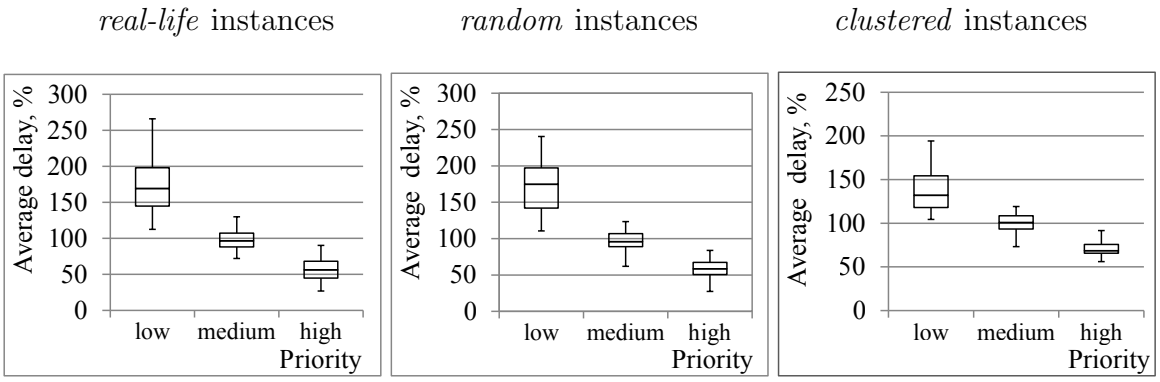


Figure 4.13: The average delay of each priority in the first run.

As expected, delays of the first run are approximately proportional to the priorities, i.e. around 60, 100 and 170 % for low, medium and large priorities, respectively. Moreover, delays of the medium priority are close to the delays of the second run. This means that delays between visits are not significantly affected by use of priorities. It was observed that, as the number of points increases, delays within one priority level deviate more. This is an expected consequence of the increasing complexity of the problem. Moreover, there was no significant difference between the performance

on the *life* and *random* scenarios. The *clustered* instances appeared to be harder to solve.

To demonstrate the effect of instance size on computational time, the IDIH-Reserve solved every instance from the *real-life*, *random* and *clustered* sets ten times. Fig. 4.14 reports the computational time averaged over all runs. The run-time increases linearly with the number of points, vehicles and visits.

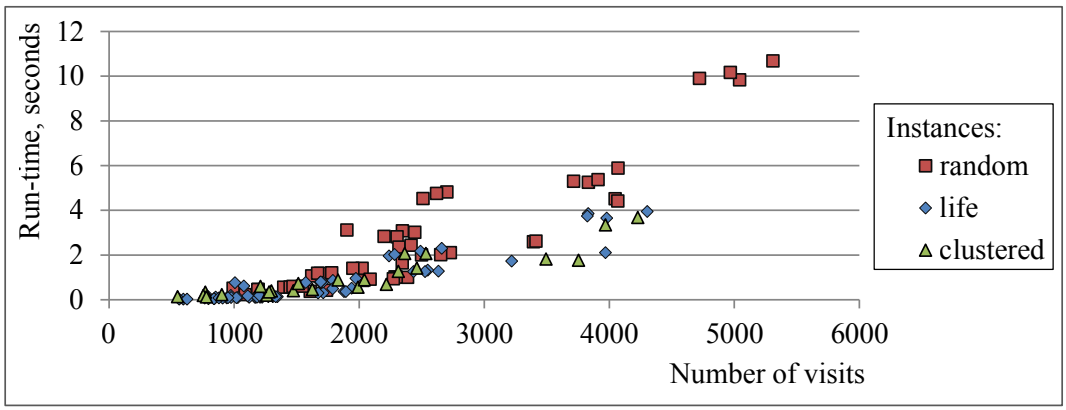


Figure 4.14: Computational time of the IDIH-Reserve.

The linear increase of the run-time comes from the algorithm's complexity that depends on the number of points, vehicles and visits. The upper bound on the number of visits is $maxCap/minTrTime$, where $maxCap$ is the maximal total capacity of all batteries of the same type, $minTrTime$ is the minimal travel time between two points. Then the complexity of the heuristic is $O(|N_p| \cdot |N_v| \cdot maxCap/minTrTime)$. According to the evaluation results, even the largest instances are solved in seconds, e.g. the instances with 800 points are solved at most in 10.69 s. These results show that the IDIH-Reserve heuristic can be used in time-critical applications.

4.3.5 Summary

In this section we describe the evaluation studies conducted for the IDIH-Reserve heuristic proposed for the CMPIDP problem. The algorithm is on average only 4 % and at most 14 % far from optimum for cases where an optimum is computable. It also achieves good results for the real-life CMPIDP instances in short time, e.g. in less than 11 s for the largest instance with 800 points. The solutions obtained by the IDIH-Reserve have visit frequency proportional to the priority level as required by the problem. Moreover, the method is linear in the number of points, vehicles and visits.

Chapter 5

Conclusion

In this thesis we described route planning problems for area monitoring: the continuous monitoring problem (CMP), CMP with inter-depot routes (CMPID) and CMPID with priorities (CMPIDP). We proposed solution methods for these problems. These problems arise in many real-life scenarios such as continuous aerial surveillance of a crime scene or an open-air concert. We supported the problems with a scenario where a fleet of unmanned aerial vehicles assists first responders by periodically providing an overview of a disaster site.

The **CMP** is a variation of the well-known NP-hard problem, vehicle routing problem. The goal of the CMP is to compute a sequence of routes for a fleet of vehicles in order to periodically visit a set of points. An optimal solution minimizes delays between consecutive visits and maximizes the number of visits within the fixed planning horizon. The vehicles can be heterogeneous, i.e., have different travel speed and energy supply on board. The energy supply can be renewed at a vehicle home station.

Due to the complexity of the problem, we proposed heuristic approaches to solve the CMP. An initial solution can be constructed with a queue-based insertion heuristic (QI) or a modified Clarke and Wright algorithm (CW). Then the solution is improved by a metaheuristic VNS. The methods were evaluated on a set of real-life benchmarks. The runtime of both construction heuristics QI and CW is less than one second even for the largest scenarios. The QI achieved better results on smaller instances, whereas the CW outperformed the QI on larger benchmarks. Since these methods are not exact algorithms, we compared their solutions with optima on instances where an optimum was computable in an admissible amount of time. For these test cases our metaheuristic obtained solutions at most 27 % worse than the optimum. We also analyzed several neighborhood structures with a new insert oper-

ator. The best operator sequence improved the initial solution cost provided by the CW on approximately 12 % on large scenarios.

The **CMPID** emerges when the vehicles are not assigned to certain stations and can recharge at any of them. By relaxing this CMP constraint, we could gain better efficiency.

To solve the CMPID problem, we developed two construction heuristics: inter-depot insertion heuristic (IDIH) and IDIH with battery reservations (IDIH-Reserve). The IDIH-Reserve achieves better results than the IDIH on most large instances due to its battery reservation policy and new parameters in the evaluation function. The IDIH-Reserve also returns near-optimal solutions with the average deviation from an optimum of only 7 %. When compared with an optimal patrolling strategy on large instances which are generated in favor of patrolling, the IDIH-Reserve solution quality is worse by only 11 %.

Solutions provided by the IDIH and IDIH-Reserve can be further improved by a VNS. The VNS differs from the version applied to the CMP, as it contains a new neighborhood operator and does not include a local search procedure. This allows an increase in the solving speed, while keeping improvements at a high level, i.e., initial solutions of the real-life scenarios are improved by 3–30 % in only 10 minutes.

The **CMPIDP** problem arises in scenarios where parts of an area have different importance, e.g., the terrain on the edge of a forest fire or locations of survivors. Apart from the problem parameters given in the CMPID, the CMPIDP includes priority information and does not have a fixed mission time. Its goal is to construct routes for periodical visitation of points that utilizes a maximal number of energy supplies and minimizes delays between visits proportional to priorities.

The IDIH-Reserve heuristic was extended to solve the CMPIDP problem. This heuristic is only 4 % on average worse than the optimum. For the larger random, clustered and real-life instances, this heuristic constructs routes with delays proportional to the priorities of points. We also showed that the performance of the IDIH-Reserve does not decrease from introducing priorities. Moreover, the algorithm is linear in the number of points, vehicles and visits and solves largest instances with 800 points in only 11 seconds.

In a real-life environment, different kinds of disturbances are very likely to occur. Since feasible solutions for the large real-world scenarios of the CMP and its extensions are found in seconds, the proposed methods can be used for quick plan adaptation to some scenario changes such as drone failure or change of priorities. The required minor modifications of the input information is described for each scenario.

We envision several directions to extend the research presented in this thesis:

1. A drawback of the approaches proposed in this thesis comes from their advantage: good coefficients settings lead to high-quality solutions, however, wrong coefficient values can decrease the efficiency. This can be solved by introducing dynamically adjusting coefficients. Their values change during the execution based on a partial solution of a problem.

2. In this work the connectivity between the drones and the base stations is assumed to be sufficient for the mission, e.g., for sending pictures to the stations. This does not hold in environments cluttered with obstacles such as urban scenarios. Future work could be to develop an approach that involves network connectivity into the planning process.

3. In this thesis we assume that each battery is used only once. In real life, batteries can be recharged at base stations and, thus, can be used multiple times. In that case where a drone leaves the battery becomes critical, as it influences further planning. The prospective future direction is to consider the use of recharged batteries.

4. A robotic system can include both aerial and ground robots. In this case, the ground robots could serve as charging stations and constantly adjust their locations to the flight paths of the UAVs. The future work is to extend the UAV route planning with moving charging stations.

5. To avoid collisions between drones, we currently check the UAVs flight paths on intersections in time and space. Assuming the drones fly in a straight line between the two points, the intersections of drone trajectories are easily computable. However, vehicles do not always follow a straight line during flight, e.g., their paths are curved in a strong wind. In this case, we envision three possible solutions: to fly at different altitudes, to cluster the area, or to forbid intersections within a certain time interval in order to compensate for real-life disturbances like wind.

6. In this thesis we considered only heterogeneous sensing drones, i.e., drones taking pictures. The future research direction is to introduce different types of services. For example, in a disaster scenario, we can distinguish between sensing, rescue and networking agents. A networking agent is a relay in an aerial

network to provide a better connectivity between the drones and the base stations. Additionally, different UAV platforms can be used for different tasks, e.g., fixed-wing UAVs for sensing and multi-copters for rescue and networking tasks.

Appendix A

COP Model

Usage: when solving the CMP, CMPID or CMPIDP, rules used for other problems should be commented out or removed.

```
% PARAMETERS (input)
int: nSt; % number of stations
int: nP; % number of points
int: nN = nSt + nP; % number of nodes

array[1..nN, 1..nN] of int: dist; % distance between two nodes

array[1..nP] of int: pr; % priority of a point for the CMPIDP
array[1..nP] of int: lvt; % time since the last visit of a point

int: nT; % number of types
array[1..nT] of int: capacity;
array[1..nT] of int: speed;
array[1..nT] of int: servTime; % e.g., to take a picture
array[1..nT] of int: time2chBat; % time to change a battery

% number of batteries of a type in a station
array[1..nSt, 1..nT] of int: nBat;

int: nV; % number of vehicles
array[1..nV] of 1..nT: vehType; % type of a vehicle

% initial remaining battery capacity of a vehicle
array[1..nV] of int: initBatCap;
array[1..nV] of 1..nN: initLoc; % initial location of a vehicle

% only for the CMP
array[1..nV] of 1..nSt: vehStation; % home station of a vehicle

int: maxNVisitsV; % maximal number of visits made by a vehicle
int: maxNVisitsP; % maximal number of visits of a point
int: mT; % misssion time for the CMP and CMPID
% DECISION VARIABLES (output)
% a sequence of nodes visited by each vehicle
array[1..nV, 1..maxNVisitsV] of var 1..nN+1: visit;
```

```
36
37 % PREDICATES
38 % firstStVisit equals the index of a visit when vehicle v visits a
39 % station for the first time. It is required for the goal function
40 predicate firstVisit2Station(var int: firstStVisit,
41     array[1..nV, 1..maxNVisitsV] of var 1..nN+1: visit, int: iV)=
42   let{array[1..maxNVisitsV] of var 1..maxNVisitsV: y} in
43   y[1] = min(maxNVisitsV,
44     1 + bool2int(not (visit[iV,1]<=nSt))*maxNVisitsV) /\
45   firstStVisit = y[maxNVisitsV] /\
46   forall(iVis in 2..maxNVisitsV)(
47     y[iVis] = min(y[iVis-1],
48       iVis + bool2int(not (visit[iV,iVis]<=nSt))*maxNVisitsV)
49   );
50
51 % Counts visits made at point iP.
52 predicate countVisits(array[1..nV,1..maxNVisitsV]of var int:visit,
53     var nSt+1..nN: iP, var int: c) =
54   c = sum(iV in 1..nV, iVis in 1..maxNVisitsV)
55     (bool2int(visit[iV,iVis] == iP));
56
57 % Sets maxTime to the maximal mission time among vehicles (CMPIDP)
58 predicate getMaxMTime(array[1..nV] of var int: vehMTime,
59     var int: maxTime) =
60   let{array[1..nV] of var int: y} in
61   y[1] = vehMTime[1] /\
62   maxTime = y[nV] /\
63   forall(iV in 2..nV)(
64     y[iV] = max(vehMTime[iV], y[iV-1])
65   );
66
67
68 % CONSTRAINTS
69 %Every vehicle must start its first route at the initial location.
70 constraint forall(iV in 1..nV)(visit[iV,1] == initLoc[iV]);
71
72 % Vehicles cannot stay at one node except for the 'dummy' node.
73 constraint
74   forall(iV in 1..nV)(
75     forall(iM in 2..maxNVisitsV)(
76       visit[iV,iM-1] == nN+1 /\ visit[iV,iM-1] == visit[iV,iM]
77       \/
78       visit[iV,iM-1] != visit[iV,iM] /\ visit[iV,iM-1] != nN+1
79     )
80   );
81 % All vehicles must finish their mission at a station.
82 constraint
83   forall(iV in 1..nV)(
84     forall(iVis in 1..maxNVisitsV-1)(
85       not (visit[iV,iVis] > nSt /\ visit[iV,iVis] <=nN) \/
86       visit[iV,iVis+1] <=nN
87     ) /\
88     (visit[iV,maxNVisitsV]<=nSt \/ visit[iV,maxNVisitsV]==nN+1)
89   );
```

```
90
91 % Flight time of a route cannot exceed battery capacity.
92 array[1..nV, 1..maxNVisitsV] of var int:usedCap;
93 constraint
94   forall(iV in 1..nV)(
95     let{ var 1..maxNVisitsV: firstStVisit} in
96     firstVisit2Station(firstStVisit,visit,iV) /\
97     usedCap[iV,firstStVisit]<=initBatCap[iV] /\
98     usedCap[iV,1] = 0 /\
99     forall(iVis in 2..maxNVisitsV)(
100      let{var bool: prevIsPoint=visit[iV,iVis-1]>nSt/\
101            visit[iV,iVis-1]<=nN,
102          var bool: isDummy = visit[iV,iVis] == nN+1} in
103      isDummy /\ usedCap[iV,iVis] = 0 \/
104      not isDummy /\
105      usedCap[iV,iVis] <= capacity[vehType[iV]] /\
106      usedCap[iV,iVis] = (usedCap[iV,iVis-1] +
107          servTime[vehType[iV]])*bool2int(prevIsPoint /\ iVis!=2)+
108          dist[visit[iV,iVis-1],visit[iV,iVis]] div
109          speed[vehType[iV]]
110    )
111  );
112
113 % For the CMP and CMPID: vehicles cannot use more batteries than
114 % there are available.
115 constraint
116   forall(iS in 1..nSt)(
117     forall(iT in 1..nT)(
118       let{var int: nChanges = sum(iV in 1..nV,
119         iVis in 2..maxNVisitsV-1)
120         (bool2int(vehType[iV] == iT /\ visit[iV,iVis] == iS /\
121           visit[iV,iVis+1] != nN+1)) } in
122       nChanges <= nBat[iS, iT]
123     )
124   );
125 % For the CMPIDP: vehicles must use all batteries.
126 constraint
127   forall(iS in 1..nSt)(
128     forall(iT in 1..nT)(
129       let{var int: nChanges = sum(iV in 1..nV,
130         iVis in 2..maxNVisitsV-1)
131         (bool2int(vehType[iV] == iT /\ visit[iV,iVis] == iS /\
132           visit[iV,iVis+1] != nN+1)
133         ) } in
134       nChanges == nBat[iS, iT]
135     )
136   );
137
138 % For the CMP: a vehicle visits only its home station and points.
139 constraint
140   forall(iV in 1..nV)(
141     forall(iM in 2..maxNVisitsV)(
142       visit[iV,iM] = vehStation[iV] \/ visit[iV,iM]>nSt
143     )
```

```
144   );
145 % Arrival time during every visit that a vehicle makes
146 array[1..nV, 1..maxNVisitsV] of var int: aTimeV;
147 constraint
148   forall(iV in 1..nV)(
149     aTimeV[iV, 1] = 0 /\
150 
151     let{ 1..nT: iT = vehType[iV]} in
152     forall(iVis in 2..maxNVisitsV)(
153       let{var bool: prevIsPoint=visit[iV,iVis-1]>nSt /\
154           visit[iV,iVis-1]<=nN,
155         var bool: isDummy = visit[iV,iVis] == nN+1} in
156       isDummy /\ aTimeV[iV, iVis] = 0 \/
157       not isDummy /\ aTimeV[iV, iVis] = aTimeV[iV, iVis-1] +
158         dist[ visit[iV,iVis-1], visit[iV, iVis] ] div
159         speed[vehType[iV]] +
160         bool2int(prevIsPoint/\iVis!=2) * servTime[iT] +
161         bool2int(not prevIsPoint/\iVis!=2)*time2chBat[iT]
162     )
163   );
164 
165 %Counts how many times a point is an initial location of a vehicle
166 array[1..nP] of var int: isInitLoc;
167 constraint
168   forall(iP in 1..nP)(
169     isInitLoc[iP]=sum(iV in 1..nV)(bool2int(initLoc[iV]==iP+nSt))
170   );
171 % Defines the total number of visits of each point.
172 array[1..nP] of var int: nVisitsP;
173 constraint
174   forall(iP in 1..nP)(
175     countVisits(visit, iP+nSt, nVisitsP[iP])
176   );
177 
178 % Derives an ordered array of arrival times of every point.
179 array[1..nP,1..maxNVisitsP] of var int:aTimeP;
180 constraint
181   forall(iP in 1..nP)(
182     % first arrival time equals to 0 if number of visits is less
183     % than the maximal possible number of visits
184     ( maxNVisitsP != nVisitsP[iP] /\ aTimeP[iP,1] = 0 \/
185       maxNVisitsP == nVisitsP[iP] /\ aTimeP[iP,1] > 0)
186     /\
187     % defines the order and which arrival times should be
188     % equal to 0
189     forall(iVisP in 2..maxNVisitsP)(
190       aTimeP[iP,iVisP] >= aTimeP[iP,iVisP-1]
191       /\
192       ( iVisP <= maxNVisitsP - nVisitsP[iP] + isInitLoc[iP] /\
193         aTimeP[iP,iVisP]=0
194         \/
195         iVisP > maxNVisitsP - nVisitsP[iP] + isInitLoc[iP] /\
196         aTimeP[iP,iVisP]>0)
197     )
```

```
198     /\
199
200     % include only arrival times of the corresponding point
201     forall(iV in 1..nV, iVis in 1..maxNVisitsV)(
202       visit[iV,iVis] == iP+nSt /\
203       exists(iVisP in 1..maxNVisitsP)
204         (aTimeP[iP,iVisP] == aTimeV[iV,iVis])
205       \/
206       visit[iV,iVis] != iP+nSt
207     )
208   );
209
210 % No two vehicles can visit a point at the same time.
211 constraint
212   forall(iV1,iV2 in 1..nV where iV1<=iV2)(
213     forall(iVis1, iVis2 in 1..maxNVisitsV where
214       not (iV1 == iV2 /\ iVis1 == iVis2))
215     (
216         aTimeV[iV1,iVis1] != aTimeV[iV2,iVis2] \/
217         visit[iV1,iVis1] != visit[iV2,iVis2] \/
218         visit[iV1,iVis1] <= nSt \/ visit[iV1,iVis1] == nN + 1
219     )
220   );
221
222 % For the CMP and CMPID: no vehicle can finish its final flight
223 % after the end of mission.
224 constraint
225   forall(iV in 1..nV)(
226     forall(iVis in 1..maxNVisitsV)( aTimeV[iV, iVis] <= mT )
227   );
228
229 % Time when a point was visited the first time.
230 array[1..nP] of var int: firstArTime;
231 constraint
232   forall(iP in 1..nP)(
233     aTimeP[iP,1] != 0 /\ firstArTime[iP] = aTimeP[iP,1]
234     \/
235     aTimeP[iP,1] == 0 /\ aTimeP[iP,maxNVisitsP] != 0 /\
236     forall(iVisP in 2..maxNVisitsP)(
237       aTimeP[iP,iVisP-1] == 0 /\ aTimeP[iP,iVisP] != 0 /\
238       firstArTime[iP] = aTimeP[iP,iVisP]
239       \/ aTimeP[iP,iVisP-1] != 0
240       \/ aTimeP[iP,iVisP] == 0)
241     \/ aTimeP[iP,maxNVisitsP] == 0 /\ firstArTime[iP] = 0
242   );
243
244
245 % Penalties for the problems with fixed mission time,CMP and CMPID
246 array[1..nP] of var int: costs;
247 constraint
248   forall(iP in 1..nP)(
249     aTimeP[iP,maxNVisitsP] == 0 /\
250     costs[iP] = (mT + lvt[iP])*(mT + lvt[iP])
251     \/
```

```
252
253     aTimeP[iP, maxNVisitsP] != 0 /\
254     let{var int: costBetweenVisits = sum(i in 1..maxNVisitsP-1)
255       ((aTimeP[iP,i+1] - aTimeP[iP,i]) *
256        (aTimeP[iP,i+1] - aTimeP[iP,i]) *
257        bool2int(aTimeP[iP, i] !=0))} in
258     costs[iP] = costBetweenVisits +
259       (mT - aTimeP[iP,maxNVisitsP])*(mT - aTimeP[iP,maxNVisitsP])+
260       (firstArTime[iP] + lvt[iP])*(firstArTime[iP] + lvt[iP])
261   );
262
263 % Penalties for the problem without fixed mission time, CMPIDP
264 % First, compute the mission time, i.e., maximal mission time
265 % among all vehicles...
266 var int: mT;
267 array[1..nV] of var int: vehMTime;
268 constraint
269   forall(iV in 1..nV)(
270     let{var int: nChanges = sum(iVis in 2..maxNVisitsV-1)
271       (bool2int(visit[iV,iVis]<=nSt /\ visit[iV,iVis+1]!=nN+1))}in
272
273     vehMTime[iV] = nChanges * (capacity[vehType[iV]] +
274       time2chBat[vehType[iV]]) + initBatCap[iV]
275   );
276
277 constraint getMaxMTime(vehMTime, mT);
278
279 % ... then compute the penalties
280 array[1..nP] of var int: costs;
281 constraint
282   forall(iP in 1..nP)(
283     aTimeP[iP,maxNVisitsP] == 0 /\
284     costs[iP] = (mT + lvt[iP])*(mT + lvt[iP])
285     \/
286     aTimeP[iP,maxNVisitsP] != 0 /\
287     let{var int: costBetweenVisits = sum(i in 1..maxNVisitsP-1)
288       ((aTimeP[iP,i+1]-aTimeP[iP,i]) *
289        (aTimeP[iP,i+1]-aTimeP[iP,i]) *
290        bool2int(aTimeP[iP,i] !=0))} in
291     costs[iP] = (costBetweenVisits +
292       (mT - aTimeP[iP,maxNVisitsP])*(mT - aTimeP[iP,maxNVisitsP])+
293       (firstArTime[iP] + lvt[iP])*(firstArTime[iP] + lvt[iP])
294       )* pr[iP]*pr[iP]
295   );
296
297 % OBJECTIVE
298 % Compute a value of the goal function.
299 var int: totalCost = sum(iP in 1..nP)(costs[iP]);
300 solve minimize totalCost;
301
302 % OUTPUT
303 % Print the corresponding assignments.
304 output [show(visit), show(totalCost)];
```

Appendix B

COP Input for the CMPIDP

```
nSt = 2; % number of stations
nP = 6; % number of points

% distance between two nodes
dist =
[|0,10,9,5,11,4,4,4,
|10,0,10,13,5,7,10,11,
|9,10,0,7,6,9,12,5,
|5,13,7,0,11,8,9,2,
|11,5,6,11,0,8,11,9,
|4,7,9,8,8,0,4,6,
|4,10,12,9,11,4,0,8,
|4,11,5,2,9,6,8,0|];

pr = [1,1,2,2,2,1]; % priority of a point
lvt = [15,6,20,11,6,9]; % time since the last visit of a point

nT = 1; % number of types
capacity = [24];
speed = [1];
servTime = [1]; % e.g., to take a picture
time2chBat = [1]; % time to change a battery

nBat = [|1|3|]; % number of batteries of a type in a station

nV = 2; % number of vehicles
vehType = [1,1]; % type of a vehicle

% initial remaining battery capacity of a vehicle
initBatCap = [6,12];
initLoc = [4,7]; % initial location of a vehicle

maxNVisitsV = 11; % maximal number of visits made by a vehicle
maxNVisitsP = 5; % maximal number of visits of a point
```

Bibliography

[1] M. Ahmadi and P. Stone. A multi-robot system for continuous area sweeping tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1724–1729, May 2006.

[2] S. Alamdari, E. Fata, and S. L. Smith. Persistent monitoring in discrete environments: Minimizing the maximum weighted latency between observations. *International Journal of Robotics Research*, 33(1):138–154, January 2014.

[3] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In A. L. C. Bazzan and S. Labidi, editors, *Advances in Artificial Intelligence SBIA*, volume 3171 of *Lecture Notes in Computer Science*, pages 474–483. Springer Berlin Heidelberg, 2004.

[4] F. Alonso, M. J. Alvarez, and J. E. Beasley. A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. *The Journal of the Operational Research Society*, 59(7):963–976, 2008.

[5] E. Angelelli and M. G. Speranza. The periodic vehicle routing problem with intermediate facilities. *European Journal of Operational Research*, 137(2):233–247, 2002.

[6] N. Azi, M. Gendreau, and J.-Y. Potvin. An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, 202(3):756–763, 2010.

[7] N. Azi, M. Gendreau, and J.-Y. Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41(0):167 – 173, 2014.

[8] R. Baldacci, E. Bartolini, A. Mingozzi, and A. Valletta. An exact algorithm for the period routing problem. *Operations Research*, 59(1):228–241, January 2011.

[9] R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

[10] M. Banerjea-Brodeur, J.-F. Cordeau, G. Laporte, and A. Lasry. Scheduling linen deliveries in a large hospital. *The Journal of the Operational Research Society*, 49(8):777–780, 1998.

[11] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, and Jr. Stewart, W. R. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, 1(1):9–32, 1995.

[12] S. Belhaiza, P. Hansen, and G. Laporte. A hybrid variable neighborhood tabu search heuristic for the vehicle routing problem with multiple time windows. *Computers & Operations Research*, 52, Part B(0):269–281, 2014. Recent advances in variable neighborhood search.

[13] J. Bellingham, M. Tillerson, A. Richards, and J. P. How. Multi-task allocation and path planning for cooperating UAVs. In S. Butenko, R. Murphey, and P. M. Pardalos, editors, *Cooperative Control: Models, Applications and Algorithms*, volume 1 of *Cooperative Systems*, pages 23–41. Springer US, 2003.

[14] A. M. Benjamin and J. E. Beasley. Metaheuristics for the waste collection vehicle routing problem with time windows, driver rest period and multiple disposal facilities. *Computers & Operations Research*, 37(12):2270–2280, 2010.

[15] F. Blakeley, B. Bozkaya, B. Cao, W. Hall, and J. Knolmajer. Optimizing periodic maintenance operations for Schindler Elevator Corporation. *Interfaces*, 33(1):67–79, 2003.

[16] O. Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, December 2003.

[17] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[18] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, Part II: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[19] G. Cannata and A. Sgorbissa. A minimalist algorithm for multirobot continuous coverage. *IEEE Transactions on Robotics*, 27(2):297–312, April 2011.

[20] X. Chen and T.-S. P. Yum. Patrol districting and routing with security level functions. In *IEEE International Conference on Systems Man and Cybernetics (SMC)*, pages 3555–3562, 2010.

[21] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[22] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[23] J.-F. Cordeau, M. Gendreau, G. Laporte, J.-Y. Potvin, and F. Semet. A guide to vehicle routing heuristics. *The Journal of the Operational Research Society*, 53(5):512–522, 2002.

[24] J.-F. Cordeau, G. Laporte, and A. Mercier. A unified tabu search heuristic for vehicle routing problems with time windows. *The Journal of the Operational Research Society*, 52(8):928–936, 2001.

[25] B. Crevier, J.-F. Cordeau, and G. Laporte. The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756–773, 2007.

[26] R. Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[27] F. Despaux and F. Robledo. Multi-trip vehicle routing problem with time windows and heterogeneous fleet. In *Seventh International Conference on Numerical Methods and Applications*, pages 100–100, 2010.

[28] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

[29] Y. Elmaliach, N. Agmon, and G. A. Kaminka. Multi-robot area patrol under frequency constraints. *Annals of Mathematics and Artificial Intelligence*, 57(3–4):293–320, 2009.

[30] S. Erdoğan and E. Miller-Hooks. A green vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 48(1):100–114, 2012.

[31] D. Favaretto, E. Moretti, and P. Pellegrini. Ant colony system for a VRP with multiple time windows and multiple visits. *Journal of Interdisciplinary Mathematics*, 10(2):263–284, 2007.

[32] A. Felipe, M. T. Ortuño, G. Righini, and G. Tirado. A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges. *Transportation Research Part E: Logistics and Transportation Review*, 71(0):111–128, 2014.

[33] P. M. Francis, K. R. Smilowitz, and M. Tzur. The period vehicle routing problem and its extensions. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 73–102. Springer US, 2008.

[34] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[35] Gecode Team. Gecode: Generic constraint development environment, 2014. Available from `http://www.gecode.org`.

[36] M. Gendreau and C. D. Tarantilis. Solving large-scale vehicle routing problems with time windows: The state-of-the-art. Technical report, CIRRELT, 2010.

[37] P. Hansen, N. Mladenović, and J. A. M. Pérez. Variable neighbourhood search: methods andapplications. *Annals of Operations Research*, 175(1):367–407, 2010.

[38] H. Hashimoto, M. Yagiura, and T. Ibaraki. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization*, 5(2):434–456, 2008. In Memory of George B. Dantzig.

[39] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[40] V. Hemmelmayr, K. F. Doerner, R. F. Hartl, and S. Rath. A heuristic solution method for node routing based solid waste collection problems. *Journal of Heuristics*, 19(2):129–156, 2013.

[41] V. C. Hemmelmayr, K. F. Doerner, and R. F. Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 195:791–802, 2009.

[42] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud. A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration. *4OR*, 12(3):235–259, 2014.

[43] V. A. Huynh, J. J. Enright, and E. Frazzoli. Persistent patrol with limited-range on-board sensors. In *49th IEEE Conference on Decision and Control (CDC)*, pages 7661–7668, 2010.

[44] L. Iocchi, L. Marchetti, and D. Nardi. Multi-robot patrolling with coordinated behaviours in realistic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2796–2801, September 2011.

[45] G. W. Kinney Jr, R. R. Hill, and J. T. Moore. Devising a quick-running heuristic for an unmanned aerial vehicle (UAV) routing system. *The Journal of the Operational Research Society*, 56(7):776–786, 2005.

[46] B. I. Kim, S. Kim, and S. Sahoo. Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, 33(12):3624–3642, 2006.

[47] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

[48] Y. Kuo. Using simulated annealing to minimize fuel consumption for the time-dependent vehicle routing problem. *Computers & Industrial Engineering*, 59(1):157–165, 2010.

[49] J. Las Fargeas, B. Hyun, P. Kabamba, and A. Girard. Persistent visitation with fuel constraints. *Procedia - Social and Behavioral Sciences*, 54:1037–1046, 2012.

[50] C. Lin, K. L. Choy, G. T. S. Ho, S. H. Chung, and H. Y. Lam. Survey of green vehicle routing problem: Past and future trends. *Expert Systems with Applications*, 41(4, Part 1):1118–1138, 2014.

[51] S. Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, December 1965.

[52] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In J. Simão Sichman, F. Bousquet, and P. Davidsson, editors, *Multi-Agent-Based Simulation II*, volume 2581 of *Lecture Notes in Computer Science*, pages 155–170. Springer Berlin Heidelberg, 2003.

[53] J.-S. Marier, C. Besse, and B. Chaib-draa. A Markov model for multiagent patrolling in continuous time. In C. S. Leung, M. Lee, and J. H. Chan, editors, *Neural Information Processing*, volume 5864 of *Lecture Notes in Computer Science*, pages 648–656. Springer Berlin Heidelberg, 2009.

[54] V. Mersheeva and G. Friedrich. Routing for continuous monitoring by multiple micro UAVs in disaster scenarios. In *European Conference on Artificial Intelligence (ECAI)*, pages 588–593, 2012.

[55] V. Mersheeva and G. Friedrich. Routing of multiple micro UAVs for rescue missions. In *Austrian Robotics Workshop (ARW)*, 2012.

[56] V. Mersheeva and G. Friedrich. Continuous monitoring problem for disaster management. In *Meeting EURO working Group on Vehicle Routing and Logistics Optimization (VeRoLog)*, 2013.

[57] V. Mersheeva and G. Friedrich. Variable neighborhood search for continuous monitoring problem with inter-depot routes. In I. J. Timm and M. Thimm, editors, *KI 2013: Advances in Artificial Intelligence*, volume 8077 of *Lecture Notes in Computer Science*, pages 106–117. Springer Berlin Heidelberg, 2013.

[58] V. Mersheeva and G. Friedrich. Multi-uav monitoring with priorities and limited energy resources. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2015.

[59] N. Mladenović. A variable neighborhood algorithm - a new metaheuristic for combinatorial optimization. In *Abstracts of papers presented at Optimization Days*, page 112, 1995.

[60] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997.

[61] R. H. Mole. The curse of unintended rounding error: A case from the vehicle scheduling literature. *The Journal of the Operational Research Society*, 34(7):607–613, 1983.

[62] C. C. Murray and M. H. Karwan. An extensible modeling framework for dynamic reassignment and rerouting in cooperative airborne operations. *Naval Research Logistics*, 57(7):634–652, 2010.

[63] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a standard CP modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer Berlin Heidelberg, 2007.

[64] N. Nigam, S. Bieniawski, I Kroo, and J. Vian. Control of multiple UAVs for persistent surveillance: Algorithm and flight test results. *IEEE Transactions on Control Systems Technology*, 20(5):1236–1251, September 2012.

[65] T. Nuortio, J. Kytöjoki, H. Niska, and O. Bräysy. Improved route planning and scheduling of waste collection and transport. *Expert Systems with Applications*, 30(2):223–232, 2006.

[66] B. Y. P. Oberlin, S. Rathinam, and S. Darbha. Today's travelling salesman problem. *IEEE Robotics & Automation Magazine*, 17(4):70–77, 2010.

[67] A. Olivera and O. Viera. Adaptive memory programming for the vehicle routing problem with multiple trips. *Computers & Operations Research*, 34(1):28–47, 2007.

[68] K. P. O'Rourke. Dynamic unmanned aerial vehicle (UAV) routing with a Java-encoded reactive tabu search metaheuristic. Master's thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, February 1999.

[69] C.-H. Park, Y.-D. Kim, and B. J. Jeong. Heuristics for determining a patrol path of an unmanned combat vehicle. *Computers & Industrial Engineering*, 63(1):150–160, 2012.

[70] C. Pippin, H. Christensen, and L. Weiss. Performance based task assignment in multi-robot patrolling. In *the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 70–76, New York, NY, USA, 2013. ACM.

[71] C. Poulet, V. Corruble, and A. E. F. Seghrouchni. Working as a team: Using social criteria in the timed patrolling problem. In *IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 933–938, 2012.

[72] M. Quaritsch, K. Kruggl, D. Wischounig-Strucl, S. Bhattacharya, M. Shah, and B. Rinner. Networked UAVs as aerial sensor network for disaster management applications. *e&i Elektrotechnik und Informationstechnik, Special Issue on Wireless Sensor Networks*, 127(3):56–63, January 2010.

[73] P. P. Repoussis, C. D. Tarantilis, and G. Ioannou. Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation*, 13(3):624–647, June 2009.

[74] A. Richards, J. Bellingham, M. Tillerson, and J. How. Coordination and control of multiple UAVs. In *the AIAA Guidance, Navigation and Control Conference*, pages 2002–4588, 2002.

[75] J. L. Ryan, T. J. Bailey, J. T. Moore, and W. B. Carlton. Unmanned aerial vehicles (UAV) route selection using reactive tabu search. *Military Operations Research*, 4(3):5–24, 1999.

[76] S. Salhi and R. J. Petch. A GA based heuristic for the vehicle routing problem with multiple trips. *Journal of Mathematical Modelling and Algorithms*, 6(4):591–613, 2007.

[77] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch. Multi-agent patrolling with reinforcement learning. In *the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1122–1129, 2004.

[78] M. W. P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

[79] M. Schneider, A. Stenger, and D. Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48(4):500–520, 2012.

[80] V. K. Shetty, M. Sudit, and R. Nagi. Priority-based assignment and routing of a fleet of unmanned combat aerial vehicles. *Computers & Operations Research*, 35(6):1813–1828, 2008. Part Special Issue: OR Applications in the Military and in Counter-Terrorism.

[81] S. L. Smith and D. Rus. Multi-robot monitoring in dynamic environments with guaranteed currency of observations. In *49th IEEE Conference on Decision and Control (CDC)*, pages 514–521, 2010.

[82] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254–265, April 1987.

[83] R. Stranders, E. Munoz De Cote, A. Rogers, and N. R. Jennings. Near-optimal continuous patrolling with teams of mobile information gathering agents. *Artificial Intelligence*, 195:63–105, February 2013.

[84] E. Stump and N. Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. In *IEEE Conference on Automation Science and Engineering (CASE)*, pages 569–575, August 2011.

[85] K. Sundar and S. Rathinam. Route planning algorithms for unmanned aerial vehicles with refueling constraints. In *American Control Conference*, pages 3266–3271, June 2012.

[86] É. D. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[87] É. D. Taillard, G. Laporte, and M. Gendreau. Vehicle routeing with multiple use of vehicles. *The Journal of the Operational Research Society*, 47(8):1065–1070, 1996.

[88] C. D. Tarantilis, E. E. Zachariadis, and C. T. Kiranoudis. A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. *INFORMS Journal on Computing*, 20(1):154–168, January 2008.

[89] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[90] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, May 2012.

[91] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21, 2013.

[92] Y. Xiao, Q. Zhao, I. Kaku, and Y. Xu. Development of a fuel consumption optimization model for the capacitated vehicle routing problem. *Computers & Operations Research*, 39(7):1419–1431, 2012.

[93] J. Yu, S. Karaman, and D. Rus. Persistent monitoring of events with stochastic arrivals at multiple stations. In *IEEE International Conference on Robotics and Automation*, 2014.